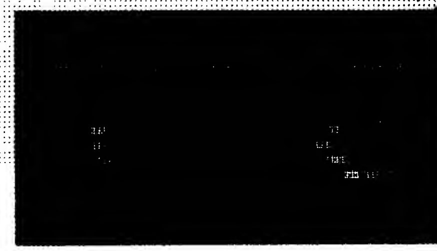
July/August 1997 … Number 1 … America)

the world of 68' micros

Support for Motorola based computer systems and microcontrollers, … operating system

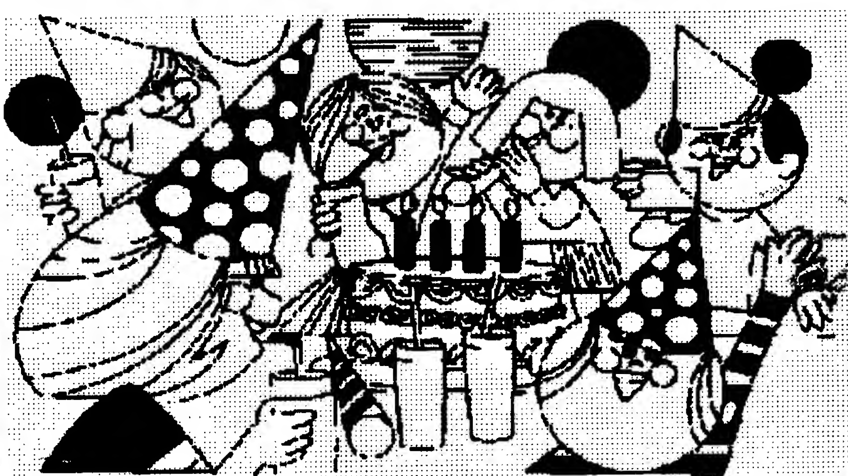
CoCo2000!



Will you still be using your CoCo in the year 2000? There should STILL be a magazine to support you (and OS-9 68K) if you are!

did anyone notice..

THIS IS THE FIRST ISSUE OF VOLUME FIVE!!!
FOUR SUCCESSFUL YEARS HAVE PASSED!!



CONTENTS

POSTMASTER:
If undeliverable return to:
FARNA Systems PB
Box 321
WR, GA 31099

If your address is incorrect, send me a postcard!

The Editor's Page

I hope I got your attention with the slight cover changes and the "CoCo 2000" headline. Basically, I am committing to producing this magazine through the year 2000. That is about three more years. Where we go beyond that is up to you, the readers and subscribers. As long as I have interesting things to print and enough subscribers to make the work worthwhile, we'll continue printing even after 2000.

What is worthwhile? We start our fifth year of publication with this issue. In this time, I have posted profits twice, the best year (nothing broke and had to be replaced!) was the last one at about $1,000. Previously I've posted maybe $250 in profits.

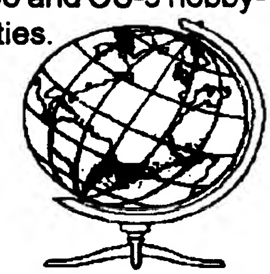
I said "profit", but I mean that very loosely! That is basically what I got out of printing the magazine. I have upgraded my computer equipment several times in order to make publishing easier, and have had to replace a few items that just quit. Of course I use my equipment for other things, so I get a little pay-back by having a little extra money to buy better equipment than I may otherwise have. But I pay myself nothing.

That profit mentioned is basically the magazine's surplus, which is held just in case something does break or I feel that something needs to be upgraded. Very rarely I will make a personal purchase from my business account that I can't write off as a business expense. That is all I get for producing the magazine. And this is mostly offset for the expenses I have at home (like the extra power, some telephone time, etc.).

The first two years had eight issues, the last two six. So that is 28 issues altogether. It takes an average of 30 hours work to produce, assemble, and mail each issue. That comes to a total of 840 hours of labor. Divide $1250 by 840 and you get $1.50 per hour.

While this is hardly enough to think about making the magazine, there are peripheral rewards. One already mentioned is that my computer equipment is a total business write-off. So I get my computer stuff basically for free. Then there are the trips to Chicago and other fests. 90% of the expenses come from my business account. And there are the contacts with subscribers, and the joy of providing something useful for others.

I have to admit, I don't print this magazine for the money! It does a little better than break even. That is enough for what is basically a hobby business. I just hope you appreciate the fact that I am willing to go to so much trouble for so little by continuing to subscribe and support others who work for as little or even less (such as Glenside CoCo Club and Ron Bull, who put on 'fests this year) to support the CoCo and OS-9 hobbyist communities.



the world of 68' micros

Publisher:
FARNA Systems PB
P.O. Box 321
Warner Robins, GA 31099-0321

Editor:
Francis (Frank) G. Swygert

Subscriptions:
US/Mexico: $24 per year
Canada: $30 per year
Overseas: $50 per year (airmail)
Back and single issues are cover price. Overseas add $3.00 one issue, $5.00 two or more for airmail delivery.

The publisher is available via e-mail dsrtfox@delphi.com

Advertising Rates:
Contact publisher. We have scales to suit every type of business. Special rates for entrepreneurs and "cottage" businesses.

Contributions:
All contributions welcome. Submission constitutes warranty on part of the author that the work is original unless otherwise specified. Publisher reserves the right to edit or reject material without explanation. Editing will be limited to corrections and fitting available space. Authors retain copyright. Submission gives publisher first publication rights and right to reprint in any form with credit given author.

General Information:
Current publication frequency is bi-monthly. Frequency and prices subject to change without notice. All opinions expressed herein are those of the individual authors, not necessarily of the publisher. No warranty as to the suitability or operation of any software or hardware modifications is given nor implied under any circumstances. Use of any information in this publication is entirely at the discretion and responsibility of the reader.

At306 Trials, Tribulations, and CORRECTIONS!!!

Good news that it wasn't the actual "last" cocofest.

That was a helpfull discussion of partitions. Now I know what those h0a, h0b descriptors are for. I think that was never explained and I never knew enough to ask. But don't they need to be included in the bootfile?? To make several partitions do you have to run format several times? What do you use in place of the /h0fmt which sets the system to bypass the write lock so you CAN format? I think I know a way around but thought you ought to tell it.

In your discussion preceeding partitioning I think there is a trouble. The sector size is 512 (bytes), not 512 kilobytes. So with map capability of 64000 bytes I think capacity would be 34M. By using cluster size 16 I think the map would address 64000 x 8192 = 524M . In other words a 256k file would need 32 clusters and really wouldn't waste a lot of space— at worst 8192 bytes.

That's interesting about running the vga adapter card on a pc to initialize it. That ought to save a gang of at306boot time in the pc mode. But I suppose yet another ROM than I have is required to take advantage of it.

The spread sheet program I was using on mm1 doesn't run on at306, and VED only sort of runs. I've had to use pc and I sure miss the multiple windows which are available on os9.

Fran Walters
72130.3067@CompuServe.COM

1) Yes, that there will be another Chicago fest is good news indeed! And don't forget about the PA fest!

2) The drivers for each partition you wish to use (h0a, h0b, etc.) must be loaded in your bootfile, as you correctly observed. h0fmt is ONLY used to format a drive with a no partitions... the entire drive is used as a single partition. When the partition drivers are used, each partition is formatted individually with the format command, there is no locking done.

*3) You are absolutely correct! I intended to write 512 BYTES, not KILOBYTES!! Sorry about all the confusion this caused. You weren't the only one to point this error out! The 64K bit map uses one bit per sector. 64K bytes * 8 bits/byte * 512 bytes/sector = 256MB per partition. This should clear up your math!!*

4) The VGA card seems to remember some settings from the PC. We're not sure exactly what transpires here, Carl is looking into it. We only know that some PC VGA cards won't work in an AT306 until they have been installed in a PC first. Has no effect on booting or the AT306 ROMs, or vice-versa.

5) What spreadsheet are you using on the MM/1? If it uses termcap, it should work on the AT306. If it is somehow MM/1 hardware specific or requires K-windows, it won't work. You may want to contact Bob VanDerPoel about VED.

1024 or 1000 = 1 Megabyte?

I believe your explanation of formatted versus unformatted capacity to be in error. An IDE hard drive with 1048 cylinders, 16 heads, and 63 sectors per track has 1048*16*63*512 bytes of storage, or 540,868,608 bytes. Hard drive manufacturers define a megabyte as one million bytes and not 1,048,576 (1024*1024) as memory manufacturers do. 540,868,608 divided by 1,048,576 gives 515.8125 megabytes of storage, which is 516 megabytes as you wrote (p8 col3 top).

I do not see that as a devious marketing ploy. Originally, the term "kilobyte" emerged because someone noticed that 2^10 is approximately 10^3 (1024 vs 1000). It is a good approximation for small amounts of bytes, but gets less accurate as the number of bytes increases.

Memory makers are tied to the approximation because the number of bits in their memory chips is a power of two. Hard drive makers are not tied to the approximation because they can make the hard drive any size and not just powers of two. If I made a hard drive that held 540 million bytes then I, too, would call it a 540MB drive and not a 516MB hard drive.

I am looking forward to the next issue and more news about the AT306.

Paul R. Santa-Maria
Ann Arbor, Michigan, USA
paul@orchard.washtenaw.cc.mi.us

You are indeed correct. Hard drive manufacturers use the estimated 1000 bytes while memory and most other peripheral manufactures use the true measurement of 1024. Some hard drive manufacturers will state the formatted size of a drive, while some give the unformatted capacity. So my explanation of why that particular drive formatted to 516MB instead of the advertised 540MB was incorrect, as your excellent math shows.

The reason hard drive manufacturers use the larger numbers, however, is marketing. The manufacturers know that the numbers will be less than advertised once the equipment is put in use. There have been many times that a novice will ask computer magazines why they bought a 540MB or 600MB drive and only get 516MB of storage space. The 540 number is more impressive, yet not totally untrue, so the marketing arm decided to use those numbers. Some go the step further and advertise something like a 600MB capacity for the same drive (unformatted... and I estimated the unformatted capacity, I'm not sure how much room sector marking actually takes, but it is a good amount on a mid-size drive). They get by with this because all three numbers (516, 540, and 600) are accurate, true numbers. But they are misleading to the public. When you put the drive in use, it is still a 516MB drive. The consumer shouldn't have to be a math whiz and familiar with all the numbers just to buy a drive! Now if it was common to actually use one of these drives unformatted or the computer reported capacity as an approximation, I could see using those "alternate" numbers. But when I buy something, I like to know what the capacity is without having to do a lot of mental math or take along a calculator. That shouldn't be much to expect!

Sundog Game Crack

John Riddle

Crack your Sundog games so they can be backed up for safety!

NOTICE: **Sundog games are still sold Rick's Computer Enterprise, Box 276, Liberty, KY 42539. This program was written so that legitimate owners will be able to backup their software for safekeeping. It is ILLEGAL to copy and sell or give away copywritten software. If you want a copy of one of the mentioned games, drop Rick's a note and ask for a price list!**

Several people I know requested info on backing up sundog games, so I whipped up a utility to automatically crack some of them so they can be backed up and stored in a safe place. Supported so far : Sinistar, Contras, Photon, Quest/Thelda, and Paladin's Legecy. Support for Kyum Gai: To Be Ninja will be added soon, and support for all Sundog games will be added eventually.

To use the crack, first make a backup of disk 1 of whichever Sundog games you want to remove the copy protection from. You'll need a backup utility which won't abort when it has read errors on track 0, because track 0 is the non-standard-formatted track on the Sundog disks. Once you have that, run suncrack.bas on the backup and it will automatically patch the appropriate bytes to remove the copy protection.

The crack works by using DSKCON (rom hook at location 55135 dec.) to read in a certain track and sector on the disk. For example, option 3 is Photon, so the program will goto line 300, and set up the track, sector and offset varibles. The offset is the position in the sector where the bytes that need to be patched reside.

After that, the subroutine at line 1000 first finds where in memory DSKCON is currently storing its data variables. Line 1010 will then do a read operation on drive 0, track 34, sector 6 into memory location 1024 (which is the text screen). I picked the text screen because it is a 'safe' memory location for this application, and you get a nice view of the sector that was patched.

Then in line 1020, the offset variable is used to get to the exact location in the sector of what needs to be patched, which is a subroutine call using JSR. That 3 byte instruction is replaced with 3 NOPs. The NOP byte code is 12 hex. The first 4 games are cracked by puting NOPs over the JSR call which reads in track 0 and checks some codes on it.

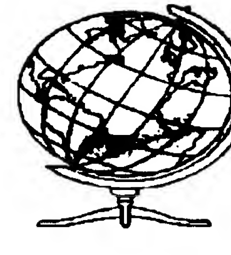
The last game is cracked by changing the execution address in the auto-start loader. This is because the copy protection is slightly different on that game in which it does the copy protection check right away. This allows for the crack to be a simple change to an execution address. The first 4 games do other things (displaying graphical sundog logo, etc) before doing the copy protection check.

Line 1030 writes the modified sector back to disk. Notice that the first poke command controls the operation (2=read and 3=write). After the patch is applied, the game will load normally since the subroutine to check the copy protected track has been blanked out.

This is not the final version of the patcher, though it will be a few months before I have a chance to perfect it (editor: this article was originally written in May, so check!). If anyone needs something clarified let me know. My address is:

312 E. Maple Road
Linthicum MD 21090
E-mail: jriddle@clark.net

```
1 'SUNCRACK V1.0 - UTILITY TO "CRACK" (NOT COPY) SOME SUNDOG GAMES
2 'BY JOHN RIDDLE MAY 22, 1997
3 'FIRST BACKUP SUNDOG DISK EXCLUDING TRACK 0
4 'THEN RUN THIS PATCH ON THE BACKUP
5 'METHODS: #1,2,3 & 4 - JSR KILL, #5 - EXEC ADDRESS CHANGE
10 F=0:CLS:WIDTH32
20 PRINT@9,"SUNCRACK V1.0"
30 PRINT:PRINT"INSERT APPROPRIATE GAME DISK   AND SELECT WHICH GAME TO CRACK"
40 PRINT:PRINT"1) SINISTAR","2) THE CONTRAS"
50 PRINT:PRINT"3) PHOTON","4) QUEST/THELDA"
60 PRINT:PRINT"5) PALADIN'S LEGACY"
70 A$=INKEY$:IF A$="" THEN 70 ELSE IF A$= "1" THEN 100 ELSE IF A$="2" THEN 200 ELSE IF A$="3" THEN 300 ELSE IF A$="4" THEN 400 ELSE IF A$="5" THEN 500 ELSE 70
100 T=30:S=12:O=&H76:GOSUB1000: GOTO 2000
200 T=3:S=11:O=&H4B:GOSUB1000: GOTO 2000
300 T=34:S=6:O=&H84:GOSUB1000: GOTO 2000
400 T=2:S=15:O=&H7E:GOSUB1000: GOTO 2000
500 T=1:S=5:O=&HC8:F=1:GOSUB1000: GOTO 2000
1000 A=PEEK(&HC006)*256+PEEK (&HC007)
1010 POKE A,2:POKEA+1,0:POKEA+2,T: POKEA+3,S:POKEA+4,4:POKEA+5,0:EXEC 55135
1020 IF F=0 THEN POKE 1024+O,&H12: POKE 1024+O+1,&H12:POKE1024+O+2, &H12
1025 IF F=1 THEN POKE 1024+O,&H32: POKE 1024+O+1,&H51
1030 POKE A,3:POKEA+1,0:POKEA+2,T: POKEA+3 ,S:POKEA+4,4:POKEA+5,0: EXEC 55135
1040 RETURN
2000 PRINT"THE GAME IS NOW CRACKED."
2010 PRINT"THE DISK CAN NOW BE COPIED USING STANDARD"
2020 PRINT"DISK BACKUP PROCEDURES."
```



From: *Dennis Bathory-Kitsz*

Hi folks! I've been hiding out in Vermont, but since it's the 10th anniversary of my company Green Mountain Micro's demise, I thought it might be time to put in an appearance here.

About 150 copies of 'Learning the 6809' (book only) remain, which I'd be happy to offer at $10 postpaid to anyone interested. If at least 10 people also want the original tapes, I'd be pleased to make up a set of those as well.

One of these days I'll tell my own tale ... amusing indeed...

Dennis Bathory-Kitsz
RD 2 Box 2770
Cox Brook Road
Northfield, Vermont 05663

<bathory@maltedmedia.com>
Malted/Media:
http://www.maltedmedia.com/

Hacking the Orchestra 90 Pak

Robert Gault

The CoCo as a Digital Sound Recorder

Background

There are many good articles in past issues of Coco magazines describing code that permits digital recording of sound via the joystick inputs. There were also some commercial programs that turned the Coco into a mini recording studio and offered sophisticated editing action. The real question is were these programs capable of good sound quality? Let's look at the theoretical software limitations of a computer with a 2MHz clock and at a hardware project with the aim of improving on these limitations.

The Coco incorporates a six bit ADC/DAC (analog to digital / digital to analog converter.) This means that there are 26 or 64 discrete signal levels that the DAC can represent. Can 64 discrete signal levels produce HiFi, mediumFi, or no fidelity? Sound levels are generally reported in units of decibels (dB). The best HiFi systems, whether analog or digital, have about a 90dB dynamic range. A 16-bit compact disc player is theoretically capable of a 96dB dynamic range. The decibel is a logarithmic number, and if related to voltages as in the Coco DAC, the formula is dB = 20 log (E1/E2). Specifically, dB = 20 log (64/1) = 36. This is pretty awful by any standard and represents fairly extreme compression and a high noise floor. Given the Coco's maximum dynamic range, it is truely amazing how good the Coco can sound when playing digitized music.

You have just read the bad news on dynamic range. Unfortunately frequency response is similarly bad. To determine the DAC frequency response limitations, we must evaluate the best code that can be written to read the DAC. The code must be symetrical in the sense that it takes exactly the same amount of time to read each of the 64 possible DAC values. If this were not true, then distortion would be added to the sound and we have more than enough already. Below is a source code fragment for reading the DAC.

```
00100 * D/A A/D CONVERTER FOR COCO3;
6BIT DAC
....
00240 DAC         EQU       $FF20
DAC located in top 6 bits 2-7
00250 KYJS        EQU       $FF00
keyboard and joystick output

....
00450
00460 * READ DAC
00470 TEST        MACRO
00480       STA         DAC
set DAC level        4 CYC
00490       LDB         KYJS
check comparator     4 CYC
00500       BMI         a@  3 CYC
00510       SUBA        #\0
reset value based on comparison 2 CYC
00520       BRA         b@          3 CYC
00530 a@ADDA            #\0
reset value based on comparison  2CYC
00540       BRN         b@
!needed to maintain symetry! 3 CYC
00550 b@          EQU       *
00560 * 16 CYCLES THROUGH EITHER PATH
00570       ENDM
00580
00590
....
00740 RECLUP        sample the DAC
00750       LDA         #32*4
preload regA with the maximum  DAC value
2 CYC
00760       TEST        16*4        16 CYC
00770       TEST        8*4         16 CYC
00780       TEST        4*4         16 CYC
00790       TEST        2*4         16 CYC
00800       TEST        1*4         16 CYC
00810       TEST        2 "0*4"  16 CYC
00820 * 98 CYCLES
00830 RECX        STA         ,X+
store in memory      6 CYC
00840 * 104 CYCLES
....
```

The only reasonable way to shorten this code would be to use a 6309 in native mode so that there are fewer clock cycles per instruction.

Keep in mind that this minimum of 104 clock cycles does not include the overhead required to check for memory overflow, or test for a keyboard stop signal.

The maximum sampling rate at 104 clock cycles per sample is thus: freq = 1.79MHz / 104 = 17212 Hz. That is not quite as good as it seems because signal theory requires a sampling rate double the highest usable frequency to prevent aliasing distortion.

The best clean frequency response possible for the Coco running in fast mode is about 8600 Hz. This is most definately low fidelity and worse than AM radio. Again, it is truely amazing just how good a well written Coco digitizer program can sound. Can we do better?!

CocoBlaster

With apologies to a commercial product of similar name, I decided to make a sound card for the Coco that would improve on the limitations of the built-in ADC. The perfect platform for this project was the Tandy Orchestra-90 Pak. This unit already has an 8-bit DAC and input/output (I/O) addressing. All that was needed was the addition of an 8-bit ADC for input.

I chose the ADC080(x) x=1,2,3,4 which used to be sold by Tandy (276-1792) and was described in the Tandy "Semiconductor Reference Guide", 1983. This unit may not be currently available but faster equivalent devices certainly are and would be better for the job. The unit has several desirable qualities: single 5v power supply, tri-state 8-bit data bus which can connected directly to the Coco, and a conversion rate capable of just barely using the Coco 2MHz clock.

The ADC080x does a conversion every 64 clock cycles. At 1.79MHz this gives a conversion rate (frequency response) of 28KHz. Thus signal theory says this chip could record a clean 14KHz signal. In fact, the program listed below can actually save data at a maximum rate of 21KHz so the clean signal is reduced to 10.5KHz. By contrast the Coco ADC, as we have just seen, is only capable of a clean 8500Hz signal. Unfortunately, the ADC080x is rated for a maximum clock of 1.46MHz and the faster Coco clock results in a conversion error 5% in amplitude. This, however, can be compensated for with a voltage offset.

So far, so good. The ADC080x can cover the 20-20000Hz human hearing range with some distortion above 11KHz. How does the dynamic range compare to the Coco DAC? Using our equation from above, dB = 20 log (28/1) = 48. That is much better but not up to HiFi standards. I would rate my project, based on listening tests, with sound quality somewhere between that of AM and FM radio.

Before leaving theory, you should understand the impact of high quality sound on storage capacity. Everything comes with a price. Sound recorded at 21KHz on a 512K RAM Coco will last about 20 seconds. Software programs running at 14KHz can store about 30 seconds of sound. About 2 1/2 double sided 40 track disks are needed to save all of this data.

Connection Requirements

The ADC080x needs both read and write signals. The ORC-90 pak has write addressing but needs read addressing installed. I did this by piggybacking a second 74LS138 chip onto the existing unit in the ORC-90 pak. A low profile IC socket was soldered onto the ORC-90 74LS138 connecting all lines except pins 5, 14, and 15.

Pin 5 is an "enable" line which will be set permanently on by tying it to ground. This will make the select lines active for both read and write periods. Pins 14 and

15 are outputs which must not interfer with the original ORC-90 lines. Pin 14 will be left unconnected. Pin 15 with go to the ADC080x chip select line, pin 1.

One more support IC will be needed to supply the read / write signals. The Coco uses a single read/write (R/W*) line with selection based on logic. The ADC080x requires its separate read and write (R pin 2, W pin 3) lines to have the same logic value. We need to decode the cartridge line 18 into separate read / write lines using and inverter for the R line. I used a 74LS02 NOR device with one input grounded as an inverter.

The data lines D0-D7 of the ADC080x must be connected to the Coco bus. I found the easiest way was to use ribbon cable to connect the ADC080x pins 18-11 (D0-D7) to the ORC-90 IC1 pins 11-19 (D0-D7.) There are several other connections which you can get from the schematic diagram below.

To access the new circuit, I chose the simple option of co-opting one of the ORC-90 RCA jacks. This made the pak a mono unit but the ADC080x is only a single channel anyway. The input to the IN+ line of the ADC080x needs to be biased at 1/2 the power supply for maximum input dynamic range. This was done with a simple resistor voltage divider and isolated from DC offset by a capacitor.

In the event that you wish to keep stereo output from the ORC-90, just put a double pole single throw switch at the RCA jack to select the original output or ADC input.

With the hardware installed, the Coco is now ready for new software.

The Software

A combination of Basic and machine code makes the new hardware function. When the program is started, the main screen is seen.

```
COCO3 512K AUDIO DIGITIZER
BY R.GAULT
SELECT YOUR FUNCTION

      (R)ECORD
      (P)LAYBACK
      (S)ET LEVELS
      (L)OOP PLAY
      (M)ONITOR    (D)ISK I/O
      (H)ARDWARE ADJUST
            BITS, FREQUENCY
            MMU BLOCKS
      (Q)UIT

SPACEBAR TO KILL FUNCTION
```

The Set Levels and Hardware Adjust screens are interesting. You can select the number of bits used per word, the sampling rate, and the length of sample captured. The level screen presents a VU meter calibrated in dB with both fast and average response. Sound quality is not altered even though the computer is busy drawing graphic paterns.

```
8-BIT VU METER
by R.Gault
!!Trim volume!!

CURRENT VALUES :
      20933 HZ  SAMPLING RATE
      8 BITS PER SAMPLE
      0 / 53 FIRST/LAST MMU BLOCK
      21.00 SEC.  SOUND SAMPLE

      SELECT (F)REQ.
             (B)ITS
             (M)MU BLOCKS
```

Source Code

A Coco3 with 512K RAM and 40 track drives is required to run the Basic and m/l routines. Change the DISKIO code for 35 track drives. On systems with Multi-Paks, the ORC-90 must be in slot #1. The following machine language source code is in EDTASM6309 format, but can easily be adapted to straight EDTASM or other editor assemblers as no 6309 specific code was used. The text and graphic outputs use the PMODE3 graphic and Lowres text screens to preserve more memory for sound. Notice how the Highres graphic screen ROM routines are used to print to the PMODE3 screen.

```
00100 * DACTMR3
00110
00120 * D/A A/D CONVERTER FOR COCO3;
8BIT DAC; 512K REQUIRED
00130 * REQUIRES ORC-90 PAK WITH 8BIT
ADC MOD
00140
00150 * USES HPRINT ROUTINE TO PRINT
TEXT ON PMODE3 SCREEN
00160   INCLUDE DISKIO
00170
00180 BFFR0     EQU     0
FIRST MMU BLOCK OF MAIN BUFFER
00190 BFFRL     EQU     $36
WOULD BE $31 TO EXCLUDE H.GRAPHICS
AREA
00200 BUFFR1    SET     $8000
00210 EBUFR1    SET     $A000
00220 BUFFR2    EQU     $2600
00230 MPI       EQU     $FF7F
00240 PIA0      EQU     $FF22
00250 PIA1      EQU     $FF23
00260 DAC8      EQU     $FF7B    ORC-
90 pak address line
00270 KYJS      EQU     $FF00    Joy-
stick output
00280 MMSLOT    SET     $FFA4
$8000-$9FFF
00290 ADCTRG    EQU     $FF7A    ADC
TRIGGER PORT
00300 ADCRED    EQU     ADCTRG ADC
READ PORT
00310 GFIRQ     EQU     $FF93    GIME
FIRQ PORT
00320 MAXLN     EQU
15+128*32+BUFFR2+50*32
00330 DELAY     EQU     20       VU
METER DECREASE DELAY
00340
00350   SETDP   $FF
00360
00370   ORG     $7000
00380
00390 ZRECRD    JMP     RECORD
00400 ZPLAY     JMP     PLAY
00410 ZLEVEL    JMP     LEVEL
00420 ZMNITR    JMP     MONITR
00430 ZCLOCK    JMP     CLOCK
00440
00450 FREQU     FDB     179      DE-
FAULT = 20KHz
00460 BITS      FCB     255      DE-
FAULT = 8 BITS
00470
00480 BUFFR0    FCB     BFFR0    DE-
FAULTS ARE ABOVE; SET FROM BASIC
DRIVER
00490 BUFFRL    FCB     BFFRL
00500
00510 DRIVES    FCB     -1,-1,-1
DRIVE NUMBERS ARE SET FROM BASIC
00520
00530 RECORD    BSR     SETUP
00540   LDA     BUFFR0
00550   STA     MMSLOT
00560 RECLUP    SYNC             WAIT
UNTIL TIMED OUT
00570   LDB     ADCRED READ ADC
00580   LDA     GFIRQ   CLEAR FIRQ
00590   STD     ADCTRG TRIGGER ADC,
SEND VALUE TO DAC
00600 RECX      STB     ,X+      SAV
ADC VALUE
00610   CMPX    #EBUFR1 END      OF
BUFFER I/O BLOCK?
00620   BNE     RECX2
00630   LDX     #BUFFR1 RESET
POINTER TO START OF BLOCK
00640   LDA     MMSLOT UPDATE MMU
VALUE
00650   ANDA    #$3F     512K COCO;
DIFFERENT VALUE NEEDED IF 1MEG COCO
00660   INCA
00670   STA     MMSLOT
00680   CMPA    BUFFRL REACHED
LAST MMU BLOCK?
00690   BNE     RECLUP
00700   BRA     RECXIT
00710 RECX2     LDA     KYJS     TEST
KEYBOARD FOR SPACEBAR
00720   BITA    #8
00730   BEQ     RECXIT
00740   BRA     RECLUP
00750 RECXIT    LDD     #$3C35
00760   STA     MMSLOT RESTORE MMU
00770   STB     $FF03    RESTART IRQ
00780   DECB
00790   STB     $FF23    SOUND OFF
00800   DECB
00810   STB     MPI      RESET MPI TO
$33
00820   LDD     #$CC
00830   STB     $FF90    RESET GIME
REGISTERS
00840   STA     GFIRQ    CLEAR GIME
FIRQ
00850   STA     $FF91    SET      FOR
SLOW TIMER
```

```
00860   STA     $FF94   CLEAR TIMER
00870   STA     $FF95   "  "
00880   LDB     FRQIMG RESET FIRQ
ROUTINE TO ROMS
00890   LDX     FRQIMG+1
00900   STB     $10F
00910   STX     $110    RESET FIRQ
00920   TFR     A,DP
00930   ANDCC   #$AF
00940   RTS
00950
00960 SETUP     ORCC    #$50
00970   LDD     #$FF7F
00980   TFR     A,DP
00990   STB     $FF02           AL-
LOW TEST FOR SPACEBAR
01000   LDA     #%11011100
COCO1,MMU,FIRQ,C-RAM,DOS
01010   STA     $FF90           ACTI-
VATE MMU
01020   LDD     #$343C  SELECT CAR-
TRIDGE AS SOUND SOURCE
01030   STA     $FF01   MUX A=0
01040   STB     $FF03   MUX B=1; CART
SOUND ON, VERT IRQ OFF
01050   LDA     #$3C
01060   STA     $FF23   SOUND ON
01070   LDA     $10F    SAVE FIRQ
PATH
01080   LDX     $110
01090   STA     FRQIMG
01100   STX     FRQIMG+1
01110   LEAX    FIRQ,PCR        IN-
STALL AN FIRQ ROUTINE EVEN IF
01120   STX     $110            IT
WON'T BE USED; IT IS USED BY LEVEL
01130   LDA     #$7E            ROU-
TINE.
01140   STA     $10F
01150   LDA     #$20
01160   STA     GFIRQ   FIRQ TIMER ON
01170   STA     $FF91   FAST CLOCK
01180   LDD     FREQU   SET BY BASIC
DRIVER
01190   STD     $FF94   SET TIMER
VALUE
01200   LDX     #BUFFR1 USED BY
RECORD AND PLAY; NOT LEVEL OR MONI-
TOR
01210   RTS
01220 FRQIMG    RMB     3
01230
01240
01250 PLAY      BSR     SETUP
01260   LDA     BUFFR0
01270   STA     MMSLOT
01280 ************************
01290 PLYLUP    SYNC            THIS
WILL MAKE PLAY RUN AT THE SAME RATE
AS
01300   LDA     GFIRQ   THE RECORD
ROUTINE.
01310 ************************
01320   LDA     ,X+
01330   ANDA    BITS    SET BY BASIC
DRIVER
01340   STA     DAC8    SEND SOUND
TO 8-BIT DAC
01350   CMPX    #EBUFR1
01360   BNE     PLYX2
01370   LDX     #BUFFR1
01380   LDA     MMSLOT
01390   ANDA    #$3F
01400   INCA
01410   STA     MMSLOT
01420   CMPA    BUFFRL
01430   BNE     PLYLUP
01440 PLYXIT    LBRA    RECXIT
01450 PLYX2     LDB     KYJS
01460   BITB    #8
01470   BEQ     PLYXIT
01480   BRA     PLYLUP
01490
01500   SETDP   0
01510
01520 * CLOCK SPEED TEST; A=$C3 AT 2MHZ
A=$E1 AT 1MHZ
01530 * CLOCK FLAG SET MEANS SLOW
1MHZ CLOCK RATE
01540
01550 CLKFLG    SET     $73FF
01560
01570 CLOCK     CLRA
01580   STA     CLKFLG,PCR
01590   PSHS    CC
01600   ORCC    #$50
01610   SYNC
01620   TST     $FF02
01630 TLOOP     INCA
01640   TST     $FF03
01650   BPL     TLOOP
01660   CMPA    #$D0
01670   BHI     CLKXIT
01680   COM     CLKFLG,PCR
01690 CLKXIT    PULS    CC,PC
01700
01710   SETDP   $FF
01720
01730
01740 * THIS ROUTINE IS TOO SLOW TO AL-
LOW THE USE OF SYNC; INSTEAD AN FIRQ
01750 * ROUTINE IS USED TO SAMPLE THE
ADC; THAT MAINTAINS HIGH QUALITY
SOUND.
01760 * THERE IS NO NEED FOR PERFECT
SYNC OF THE VU METER WITH INCOMING
SOUND.
01770
01780 LEVEL     LBSR    SETUP
01790   CLR     GFIRQ   DON'T USE
TIMER
01800   CLR     MPI     POINT TO ORC-
90 PAK IN SLOT #1
01810   LDA     #$3D
01820   STA     PIA1
01830   STA     ADCTRG INITIATE ADC
HARDWARE
01840   LBSR    LABEL
01850   LDD     #MAXLN
01860   STD     MAXV
01870   LDB     #DELAY
01880   STB     WAIT
01890 LEVEL2    LDX
#BUFFR2+15+50*32
01900   ANDCC   #%10111111      EN-
GAGE FIRQ
01910 LEVLUP    LDB     KYJS
01920 LEVX      BITB    #8
01930   LBEQ    RECXIT
01940   LDA     ADCIMG
01950   SUBA    #6      USED TO COR-
RECT HIGH SPEED ADC ERROR
01960   TSTA            COMPENSATE
FOR CONSTANT 2.5V OFFSET
01970   BPL     NORM
01980   NEGA
01990 NORM      LDB     #32     32
BYTES PER GRAPHIC SCREEN LINE
02000   MUL
02010 * ANDB    #256-(BYTES:PER:LINE)
ONLY NEEDED IF REG.B NOT = BYTES
02020   LEAY    D,X     POINT TO COR-
RECT BYTE
02030   PSHS    Y       SAVE REG.Y
FOR COMPARISONS
02040   CMPY    MAXV    PREVIOUS
HIGH VALUE
02050   BHS     LOWER   GRAPH IN-
VERTS DIRECTION
02060   STY     MAXV    SAVE NEW
HIGH VALUE
02070 LOWER     LDY     MAXV
02080   CMPY    #MAXLN
02090   BEQ     SET
02100   LDA     #%10101010
BACKGROUND COLOR
02110   DEC     WAIT    USED TO GIVE
FAST RISE, SLOW FALL TO PEAK IND.
02120   BNE     CLRLP2
02130   LDB     #DELAY  RESET SLOW-
DOWN COUNTER
02140   STB     WAIT
02150   CLR     ,Y      ERASE THE
MAX PEAK INDICATOR
02160   LEAY    32,Y    UPDATE
POINTER BY ONE LINE
02170   STY     MAXV    SAVE NEW
MAX VALUE POINTER
02180   CMPY    #MAXLN  DID WE REACH
THE BASE LINE?
02190   BEQ     SET
02200 CLRLP2    STA     ,Y      SET
PEAK INDICATOR TO FOREGROUND COLOR
02210   LEAY    32,Y    NEXT LINE
02220   CMPY    ,S      REACHED THE
CURRENT VALUE YET?
02230   BHI     CYCLE   GONE PAST IT
02240   BEQ     SET     IF PEAK NOW
SAME AS CURRENT VALUE
02250 * CLEAR A GAP BETWEEN THE OLD
MAXIMUM INDICATOR AND CURRENT VALUE
02260 CLRLUP    CLR     ,Y      SET
TO BACKGROUND COLOR
02270   LEAY    32,Y    NEXT LINE
02280   CMPY    ,S      REACHED THE
CURRENT VALUE YET?
02290   BLO     CLRLUP
02300 * FILL AN INDICATOR COLUMN
THROUGH MAXLINE VALUE
02310 SET       LDA     #$FF
02320 SETLUP    STA     ,Y      SET
TO FOREGROUND COLOR
02330   LEAY    32,Y    NEXT LINE
02340   CMPY    #MAXLN  REACHED THE
BASE LINE?
02350   BLS     SETLUP  IF NO OR THE
SAME
02360 CYCLE     LEAS    2,S     RE-
SET SYSTEM STACK "PULS REG.Y"
02370   BRA     LEVLUP
02380
02390 ADCIMG    RMB     1       A TO
D CONVERTOR IMAGE; FILLED BY FIRQ
02400 MAXV      RMB     2       AD-
DRESS OF MAX VOLTAGE LEVEL
02410 WAIT      RMB     1       DE-
LAY VALUE FOR MAXV DECREASE RATE
02420
02430 * FIRQ: THIS ROUTINE READS ADC,
SENDS VALUE TO DAC, STORES VALUE
02440 *         IN ADC-IMAGE, CLEARS GIME
FIRQ, TRIGGERS ADC.
02450 * REGISTERS CHANGED: NONE
02460
02470 FIRQ      STD     REGIMG
```

```
02480     LDB     ADCRED  READ THE ADC
02490     LDA     PIA0    CLEAR CART
FIRQ
02500     STD     ADCTRG  TRIGGER THE
ADC; SEND SOUND TO DAC
02510     STB     ADCIMG  UPDATE
IMAGE
02520     LDD     REGIMG
02530     RTI
02540 REGIMG  RMB     2
02550
02560 LABEL   PSHS    X,Y,U
02570     LDU     #$F09D
HSCREEN ASCII SET
02580     LDX     #BUFFR2+4+46*32
02590     LEAY    MES1,PCR
02600     BSR     PRINT
02610     LDX     #BUFFR2+4+71*32
02620     LEAY    MES2,PCR
02630     BSR     PRINT
02640     LDX     #BUFFR2+4+97*32
02650     LEAY    <MES3,PCR
02660     BSR     PRINT
02670     LDX     #BUFFR2+4+123*32
02680     LEAY    <MES4,PCR
02690     BSR     PRINT
02700     LDX     #BUFFR2+4+148*32
02710     LEAY    <MES5,PCR
02720     BSR     PRINT
02730     LDX     #BUFFR2+4+174*32
02740     LEAY    <MES6,PCR
02750     BSR     PRINT
02760     LDX     #BUFFR2
02770     LEAY    <MES7,PCR
02780     BSR     PRINT
02790     LDX     #BUFFR2+9*32
02800     LEAY    MES8,PCR
02810     BSR     PRINT
02820     LDX     #BUFFR2+26*32
02830     LEAY    MES9,PCR
02840     BSR     PRINT
02850     PULS    X,Y,U,PC
02860
02870 * THIS ROUTINE USES THE GRAPH-
ICS PRINT ROUTINE FROM ROM.
02880
02890
02900 PRINT   CLRA
02910     TFR     A,DP
02920     LDA     ,Y+     GET A
CHARACTER FROM MESSAGE
02930     BEQ     PRTS
02940     PSHS    X,Y
02950     SUBA    #$20    INDEX REG.A
TO ASCII SET; SUB. SPACE
02960     LDB     #8      8 LINES PER
CHARACTER
02970     MUL
02980     LEAY    D,U     POINT TO
CHARACTER IN GRAPHICS SET
02990     LDB     #%01010101
03000     STB     $B5
03010     LDB     #8      8 BYTES PER
CHARACTER
03020 LPLOOP  LDA     ,Y+     GET
PIXELS
03030     PSHS    B
03040     JSR     $F01A   USE COCO3
HIGH RES. GRAPHICS PRINT TEXT
03050     LEAX    30,X    MOVE
SCREEN POINTER TO NEXT LINE
03060     PULS    B
03070     DECB            UPDATE
COUNTER
03080     BNE     LPLOOP
03090     PULS    X,Y
03100     LEAX    2,X     MOVE RIGHT
ONE SPACE
03110     BRA     PRINT
03120 PRTS    LDA     #$FF
03130     TFR     A,DP
03140     RTS
03150
03160 * LABELS FOR THE VU METER
GRAHICS SCREEN
03170
03180 MES1    FCC     /+ 2DB/
03190     FCB     0
03200 MES2    FCC     /+ 0/
03210     FCB     0
03220 MES3    FCC     /- 2/
03230     FCB     0
03240 MES4    FCC     /- 4/
03250     FCB     0
03260 MES5    FCC     /-12/
03270     FCB     0
03280 MES6    FCC     /-46DB/
03290     FCB     0
03300 MES7    FCC     /8-BIT VU
METER/
03310     FCB     0
03320 MES8    FCC     / by R.Gault/
03330     FCB     0
03340 MES9    FCC     /!!Trim volume!!/
03350     FCB     0
03360
03370
03380 MONITR  LBSR    SETUP
03390 MMLUP   SYNC
03400     LDB     ADCRED  READ THE
ADC
03410     LDA     GFIRQ   CLEAR THE
FIRQ
03420     ANDB    BITS    BITS SET BY
BASIC DRIVER
03430     STD     ADCTRG  TRIGGER ADC,
SEND VALUE TO DAC
03440     ANDB    #%11111100
03450     STB     $FF20
03460     LDB     KYJS    TEST KEY-
BOARD FOR SPACEBAR
03470     BITB    #8
03480     LBEQ    RECXIT
03490     BITB    #%01000000
SHIFT KEYS
03500     BNE     MMLUP
03510 KLOOP   LDB     KYJS
DEBOUNCE THE SHIFT KEYS
03520     BITB    #%01000000
03530     BEQ     KLOOP
03540     COM     $530
03550     LDA     $FF03
03560     EORA    #8      TOGGLE
BETWEEN COCO DAC AND ORC-90 PAK
03570     STA     $FF03
03580     BRA     MMLUP
03590
03600     END
```

```
00100 * DISKIO: INCLUDE FILE FOR DIGIT; 8
BIT DAC
00110
00120 BUFFR0  SET     $7012
00130 BUFFRL  SET     $7013
00140 BUFFR1  SET     $4000
00150 EBUFR1  SET     $8000
00160 MMSLOT  SET     $FFA2
00170 DSKCON  SET     $C004
00180 CLKFLG  SET     $73FF
00190
00200     ORG     $7400
00210 SAVE    PSHS    CC      SAVE
INTERRUPT SETTINGS
00220     ORCC    #$50    KILL INTER-
RUPTS
00230     TST     CLKFLG,PCR      TEST
COCO CLOCK SPEED AT START OF PRGM
00240     BEQ     A@      DON'T SPEED
UP CLOCK IF SYSTEM CAN'T HACK IT
00250     STA     $FFD9
00260 A@      LDA     #3
WRITE COMMAND
00270     LDB     $7014   GET DRIVE
NUMBER
00280     STD     $EA     TELL DSKCON
00290     CLRB
00300     STB     $EC     START AT
TRACK 0
00310     INCB
00320     STB     $ED     START AT
SECTOR 1
00330     LDA     BUFFR0  GET START-
ING MMU BLOCK #
00340     STA     MMSLOT
00350 B@      LDX     #32
TRANSFER 32 SECTORS
00360     LDY     #BUFFR1
00370     STY     $EE     TELL DSKCON
WHERE TO READ DATA
00380 C@      JSR     [DSKCON]
00390     TST     $F0     ANY ERRORS?
00400     BNE     SAVXIT  YES?, THEN
QUIT
00410     BSR     TSUDAT  UPDATE
POINTERS
00420     INC     $EE
00430     LEAX    -1,X
00440     BNE     C@
00450     LDA     MMSLOT  RESET THE
DATA MMU BLOCK
00460     ANDA    #$3F
00470     INCA
00480     STA     MMSLOT
00490     CMPA    BUFFRL
00500     BNE     B@
00510 SAVXIT  LDD     #$3A00
00520     STA     MMSLOT  RESET TO
SYSTEM MMU BLOCK
00530     TST     CLKFLG,PCR
00540     BNE     D@
00550     STA     $FFD8
00560 D@      STB     $FF40   STOP
DISK DRIVE
00570     PULS    CC,PC
00580
00590 TSUDAT  INC     $ED     UP-
DATE DISK SECTOR
00600     LDA     $ED
00610     CMPA    #19     INTO THE
NEXT TRACK?
00620     BNE     A@
00630     LDA     #1      YES?, THEN
RESET SECTOR NUMBER
00640     STA     $ED
00650     INC     $EC     INCREASE
TRACK #
00660     LDA     $EC
00670     CMPA    #40     END OF 40
TRACK DISK?
00680     BNE     A@
00690     CLR     $EC     RESET TRACK #
00700     LDA     $EB     AT END OF
DRIVE SEQUENCE?
00710     BMI     D@
```

```
00720   BNE     B@
00730   LDA     $7015   GET SECOND DRIVE NUMBER
00740   BRA     C@
00750 B@        LDA     $7016   GET THIRD DRIVE NUMBER
00760 C@        STA     $EB
00770 A@        RTS             RETURN FROM UPDATES
00780 D@        LEAS    2,S     FINISHED SO RETURN TO MAIN PROGRAM
00790   BRA     SAVXIT
00800
00810   ORG     $74B0
00820
00830 READ      PSHS    CC
00840   ORCC    #$50
00850   TST     CLKFLG,PCR
00860   BEQ     A@
00870   STA     $FFD9
00880 A@        LDA     #2
00890   LDB     $7014
00900   STD     $EA
00910   CLRB
00920   STB     $EC
00930   INCB
00940   STB     $ED
00950   LDA     BUFFR0
00960   STA     MMSLOT
00970 B@        LDY     #BUFFR1
00980   LDX     #32
00990   STY     $EE
01000 C@        JSR     [DSKCON]
01010   TST     $F0
01020   LBNE    SAVXIT
01030   LBSR    TSUDAT
01040   INC     $EE
01050   LEAX    -1,X
01060   BNE     C@
01070   LDA     MMSLOT
01080   ANDA    #$3F
01090   INCA
01100   STA     MMSLOT
01110   CMPA    BUFFRL
01120   BNE     B@
01130   LBRA    SAVXIT
```

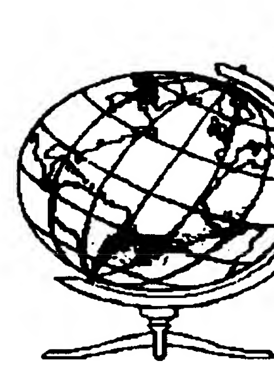
```
10 FORI=0TO2:READDR:POKE &H7014+I,DR: NEXT:GOTO70
20 DATA 0,2,255
30 WIDTH32:PRINT"YOU MUST EDIT LINE 1 TO INDICATEYOUR DRIVE SYSTEM. IF YOU HAVE DOUBLE SIDED DRIVES, THE FIRST TWO NUMBERS MUST BE THOSE FOR THE FRONT AND BACK SIDES OF THE PRIMARY DRIVE."
40 PRINT"IF YOU HAVE SINGLE SIDED DRIVES,INDICATE WHICH DRIVES SHOULD BE USED."
50 PRINT"EX.1 SINGLE DRIVE, SINGLE SIDED":PRINT"DATA 1,-1,-1"
60 PRINT"EX.2 TWO DRIVES, DOUBLE SIDED":PRINT"DATA 0,2,1":END
70 PCLEAR8:LOADM"DACTMR3": POKE &HFF40,0
80 CLOCK=&H700C:RECORD=&H7000: PLAYBACK=&H7003:LEVEL=&H7006: DSAVE =&H7400:DREAD=&H74B0: MONITOR=&H7009: P=1:EXEC CLOCK
90 BB=8:FREQUENCY=&H700F:BITS=&H7011: POKE FREQUENCY,0:POKE FREQUENCY+1,171:POKE BITS,255:' DEFAULTS ARE 20KHZ AND 8 BITS
100 TMR=2.571428571:DEF FNF1(X)= 20933*(PEEK(&H7013)-PEEK(&H7012))/TMR/RATE
110 DEF FNF2(X)=INT(20933*(53-PEEK(&H7012))/TMR/RATE+.5)
120 GOSUB410
130 POKE65497,0:WIDTH32:ONBRK GOTO500
140 PALETTE12,63:PALETTE13,0: PALETTE4, 63:PALETTE5,00:PALETTE6,11: PALETTE7,36
150 CLS:PRINTTAB(3)"COCO3 512K AUDIO DIGITIZER":PRINTTAB(10)"BY R.GAULT"
160 PRINTTAB(6)"SELECT YOUR FUNCTION"
170 PRINT:PRINTTAB(10)"(R)ECORD": PRINTTAB(10)"(P)LAYBACK":PRINTTAB(10) "(S)ET LEVELS
180 PRINTTAB(10)"(L)OOP PLAY"
190 PRINTTAB(5)"(M)ONITOR";TAB(18) "(D)ISK I/O"
200 PRINTTAB(10)"(H)ARDWARE ADJUST": PRINTTAB(13)"BITS, FREQUENCY"
210 PRINTTAB(13)"MMU BLOCKS"
220 PRINTTAB(10)"(Q)UIT"
230 PRINT:PRINTTAB(3)"SPACEBAR TO KILL FUNCTION"
240 GOSUB600:A=INSTR(1,"RPSLQDMH", A$):ONA+1 GOTO 240,250,260,270,420, 500, 700,460,300
250 CLS:GOSUB410:PRINTTAB(5) "RECORDING IN PROGRESS....": PRINT: PRINT TAB (5)" USING";BB; "BITS AT";RATE;"HZ.": EXEC RECORD:GOTO130
260 GOSUB410:GOSUB590:EXEC PLAYBACK: GOTO130
270 CLS
280 PRINT:PRINT"LEVEL CHECK ....": GOSUB510: EXEC LEVEL:RGB
290 GOTO130
300 CLS:GOSUB410:PRINT"CURRENT VALUES:":PRINTRATE;"HZ SAMPLING RATE":PRINTBB;"BITS PER SAMPLE": PRINTCB;"/";CL-1"FIRST/LAST MMU BLOCK":PRINTUSING"##.##";CR;:PRINT" SEC. SOUND SAMPLE"
310 PRINT:PRINT"SELECT (F)REQ.":PRINT TAB(7)"(B)ITS":PRINTTAB(7)"(M)MU BLOCKS"
320 GOSUB600:A=INSTR(1,"FBM ",A$): ONA +1 GOTO 320,360,330,610,130
330 PRINT:PRINT"ENTER NEW VALUE OF BITS PER":PRINT"SAMPLE (1-8)"
340 GOSUB600:BB=VAL(A$):IFBB<1 OR BB>8 THEN340
350 POKE BITS,256-2^(8-BB):GOTO300
360 PRINT:PRINT"ENTER NEW SAMPLING RATE":INPUT"(5000-20,000 HZ)";RATE: IFRATE=0THENGOTO300
370 IFRATE<5000 THENRATE=5000
380 IFRATE>20933THENRATE=20933
390 RATE=INT(3579545/RATE+.5)
400 AF=INT(RATE/256):BF=RATE-256*AF:POKE FREQUENCY,AF:POKE FREQUENCY+1,BF:GOTO300
410 RATE=INT(3579545/(256*PEEK(FREQUENCY)+PEEK(FREQUENCY+1)) +.5):CR=FNF1(X):CB=PEEK(&H7012):CL= PEEK(&H7013):RETURN
420 ONBRKGOTO130:GOSUB590
430 PRINT:PRINTTAB(5)"SPACEBAR TO RESTART"
440 PRINTTAB(5)"SPACEBAR+BREAK FOR MENU"
450 EXEC PLYBACK:GOTO450
460 CLS:PRINTTAB(10)"SOUND MONITOR":PRINT
470 GOSUB410:PRINTTAB(0)"CURRENTLY USING";BB;"BITS,";RATE;"HZ"
480 PRINT:PRINT:PRINT"SHIFT KEYS TOGGLE BETWEEN THE  COCO DAC AND THE ORC-90 PAK":PRINT"SET HARDWARE TO 6 BITS FOR TRUE":PRINT"COMPARISON"
490 EXEC MONITOR:GOTO130
500 RGB:WIDTH80:END
510 PMODE3,5:PCLS1:SCREEN1,1
520 LINE(116,50)-(118,50),PSET
530 LINE(116,50)-(116,76),PSET
540 COLOR2,0
550 LINE(116,76)-(116,178),PSET
560 FORY=75.6TO178STEP25.6
570 LINE(116,Y)-(118,Y),PSET:NEXT
580 RETURN
590 CLS:PRINTTAB(5)"PLAYBACK IN PROGRESS...":PRINT:PRINTTAB(5)"USING";BB;"BITS AT";RATE;"HZ.":RETURN
600 A$=INKEY$:IFA$=""THEN600ELSE RETURN
610 PRINT"<ENTER> RETAINS CURRENT VALUE"
620 INPUT"ENTER STARTING BLOCK #; MAX.=53 ";A$:IFA$=""THEN650ELSEBL=VAL(A$)
630 IFBL>53THENPRINT"INVALID MMU NUMBER":GOTO300
640 POKE&H7012,BL
650 CR=FNF2(X):PRINT:PRINT"ENTER LENGTH OF SOUND SEGMENT IN SECONDS (MAX.=";CR;:LINEINPUT") >>";A$:IFA$=""THEN300 ELSELG=VAL(A$)
660 IFLG<.6THENLG=.6 ELSEIFLG>CR THENLG=CR
670 POKE &H7013,INT(2.571238571*LG+.5)+BL:GOTO300
700 POKE65496,0:CLS:PRINTTAB(10)"DISK I/O":PRINT"SELECT (R)EAD (S)AVE"
710 GOSUB600:A=INSTR(1,"RS",A$):ONA+1 GOTO 130,720,730
720 EXEC DREAD:GOTO130
730 EXEC DSAVE:GOTO130
```

Caveats

The disk I/O routines are primative. Data is dumped directly to disk in a format where there is no file name and the entire disk is used for sound data. Make sure that the disks have been formatted to 40 tracks (you can use OS-9) and that there is no valuable data on the disks!

If there are questions, I can be reached via the magazine or through the internet at: robert.gault@worldnet.att.net

Happy hacking!!



RS-232 Communications with CoCo OS-9

Jim Cross

Exploring transmission speed problems and lock-ups

There are numerous factors that determine how fast a data communications rate a CoCo can handle. These include the RS-232 hardware; specific OS-9 modules: kernel, IOMan, clock driver, device driver; data buffer size; and the terminal program used. All of these factors combine to form the system limitation on the receiving data rate for any given CoCo. Since there are numerous editions of the modules and programs in circulation, that rate is different for each differently configured CoCo.

Last Christmas I added a 14.4Kbps modem to my CoCo. I faced the problem of determining how fast a receiving data rate my Coco could keep up with under OS-9. Trial and error is one approach, but I found early on that approach on outside lines is inconclusive. During heavy net traffic, my service provider slowed transmissions so that the higher data rates seem to work fine. During lighter periods, however, I'd lose data while set at the same rate that had worked last time.

I decided to try to measure how long it took my CoCo to process a received character. My DISTO 4-N-1 uses a 6551 ACIA which can only buffer 1 character, the CPU must read character #n before the 6551 ACIA finishes receiving character #n+1 or an overflow occurs and character #n is lost (overwritten by character #n+1). Note that this method of measurement only works if the ACIA is the only device using the CART/ interrupt signal during the measurement period. Also, be sure no other program tasks (except your terminal program) are running or they might affect your results.

I connected a 100 Mhz oscilloscope to the CART/ interrupt line coming from the 4-N-1 at pin 8 of CoCo cartridge interface connector. Triggering on the CART/ signal going low (signifying a character has just been received), I observed the worst case time delay until the CART/ signal returned high (signifying that the CPU has read the character from the ACIA). It's true that events other within the ACIA besides character_received can cause an interrupt, but they are not normally encountered during data reception. Using my favorite terminal program (Supercomm), I logged onto my local provider using 8-1-None and dumped all of my email to the CoCo's screen while observing the CART/ signal on the oscilloscope. Saving the data to disk may take more system time and thus can increase the worst case delay. On my system (6809) running Supercomm under OS-9 (w/ Alan DeKok's TuneUp): best case time was approximately 80 usec. = 12,500 cps (125,000 bps) but worst case time was about 1.48 msec. = 675 cps (6750 bps)

This showed me that unless I changed something in my system to reduce my CoCo's worst case response time under OS-9, I could never reliably receive at rates of 960 cps (9600 bps) or above no matter how large a receiving buffer I had (I currently use 2K buffers for receive and transmit).

I tried RTS/DTR (CTS rewired) handshaking using a new RTS circuit reputed to be from Sockmaster. This circuit stops data flow from the modem after every received character and restarts flow when the character is read by the CPU. My PM144MT II modem signals the remote modem to stop flow whenever RTS goes low; I presume other modems do also. While the new RTS circuit prevented receiver overruns, the stop-to-restart delay between my modem and the remote modem was 10 msec a any data rate, that limited my effective data rate to only 100 cps (1000 bps) regardless of the ACIA setting. One step forward, two steps back.

The CoCo uses Asynchronos data transmission, the data rate standards for which define the bit times within a character, but there is no limit to the amount of time that can occur between the end of one character and the beginning of the next unless the software imposes a timeout. This undefined intercharacter delay is why the CoCo can transmit at 19.2 Kbps but can't receive that fast under OS-9.

Also, one thing many OS-9 users may not realize is that if they are communicating with 7-1-Even and the parameters xon and xoff are set to $11 and $13 respectivel in their communications device descriptor (i.e.— /t2) with most serial device drivers they are using XON/XOFF flow control. Both parameters must be set to 00 to disable XON/XOFF flow control. Unless one knows what to look for, the XON/XOFF operation appears transparent and can hide the limiting effects of too small of a receive buffer.

If a reader has comments or speculation on this subject, I can be reached via internet at:

ac999@detroit.freenet.org

If you don't have internet access, write the editor and your comments will be passed along.

Embedded programmer

Paul K. McKneely

Exploring 68000 architecture

figure 1

```
                     Data Register
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+++-+-+-+-+-+-+-+++-+-+-+-+-+-+-+
|                    |         |         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+++-+-+-+-+-+-+-+++-+-+-+-+-+-+-+
                               <—— 8 bit —>
                     <——————— 16 bit ———————>
<—————————————————————— 32 bit ——————————————————————>
```

figure 2

```
                     Address Register
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+++-+-+-+-+-+-+-+++-+-+-+-+-+-+-+
|                    |         |         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+++-+-+-+-+-+-+-+++-+-+-+-+-+-+-+
                     <——————— 16 bit ———————>
<—————————————————————— 32 bit ——————————————————————>
```

figure 3

```
                     Status Register
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          |T  S   | IPM |   |X N Z V C|
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          <- System Byte -><- User Byte —>
```

In this article we will explore the architecture of the Motorola 68K family and begin to lay the foundation for a very powerful operating system. I will give a short review of its programmer model and then we'll discuss the processor's powerup process.

Programmer's Model

All members of the 68K family are modeled after the original 68000. It came in one of those large 64-pin DIP (Dual Inline Packages) that took up a lot of board space. It was quite a large part compared to the surface mount 68EC000 made today.

Although the chip only has 16 data lines and 24 address lines, from the programmer's point of view, it is a 32-bit architecture. This was very forward looking on the part of Motorola because even at the time they started to design the chip they did not have the ability to put that much circuitry onto one piece of silicon. I remember literature back then said it had 68,000 transistors. That number was easy to remember because it was its name.

There are 8 general-purpose Data Registers (named D0 thru D7) and 8 general-purpose Address Registers (named A0 thru A7). All are 32-bits in width. Operations on data are performed in the low-order portion of the Data Registers and can be 8-bits, 16-bits, or 32-bits in size. The upper portions of the registers are generally not altered in 8 and 16-bit operations. When operations store their results in a Data Register, the flags in the Status Register (discussed below) generally reflect the stored value. This is even true of MOVE operations, something that makes the 68K different from other processors such as the 80X86 where MOVEs don't alter the flags. A generalized Data Register is shown in table 1.

Address registers are generally used to contain and manipulate pointers. They are also very efficient at handling signed integers, although operations that store values into them usually don't affect the flags. The instructions that test and compare addresses are the exceptions to this rule. Addresses are either 16-bits (Short Addresses) or 32-bits (Long Addresses) in size. 16-bit operations always affect the whole register because the high order bit of the Short Address (bit 15) is always propagated through the upper half of the register. This is called "Sign Extension".

Both kinds of addresses (Short and Long) point into the same 32-bit address space. When you use a Short Address, you can only reach a total of 64 KBytes of address space. But because of the sign extension, it is divided into two halves. The first 32K occupies the very beginning of memory ($00000000 thru $00007FFF) and the last 32K occupies the very end of memory ($FFFF8000 thru $FFFFFFFF). The dollar sign indicates base 16 or hexadecimal in the Motorola world. Short Addressing is very efficient and we will be taking full advantage of it.

The A7 register doubles as the Stack Pointer and is sometimes referred to as the SP. All instructions that implicitly involve the Stack Pointer make use of A7. There are actually two A7 registers specifically referred to as the SSP (Supervisor Stack Pointer) and USP (User Stack Pointer). Which one is active depends on the state of the Supervisor Bit in the Status Register. A generalized Address Register is shown in figure 2. In addition, there is a 16-bit Status Register (SR). This is divided into an 8-bit System Byte and an 8-bit User Byte. This is diagramed in figure 3.

The System Byte contains three elements. Most important is the Supervisor Bit (S bit 13 above). When set (value is 1), the processor is said to be in Supervisor Mode. When clear (value is 0) it is said to be in User Mode. Supervisor Mode is more powerful because there are a set of Privileged Instructions that can be executed when a program is running in that mode. The processor will not allow these instructions to be executed when in User Mode. This bit also controls which of the two A7 registers are active.

HawkSoft

28456 S.R. 2, New Carlisle, IN 46552
219-654-7080 eves & ends MO, Check, COD; US Funds
Shipping included for US, Canada, & Mexico

MM/1 Products (OS-9/68000)

CDF $50.00 - CD-ROM File Manager! Unlock a wealth of files on CD with the MM/1! Read most text and some graphics from MS-DOS type CDs.

VCDP $50.00 - New Virtual CD Player allows you to play audio CDs on your MM/1! Graphical interface emulates a physical CD player. Requires SCSI interface and NEC CD-ROM drive.

KLOCK $20.00 - Optional Cuckoo on the hour and half hour!! Continuously displays the digital time and date on the /term screen or on all open screens. Requires I/O board, I/O cable, audio cable, and speakers.

WAVES vr 1.5 $30.00 - Now supports 8SVX and WAV files. Allows you to save and play all or any part of a sound file. Merge files or split into pieces. Record, edit, and save files; change playback/record speed. Convert mono to stereo and vice-versa! Record and play requires I/O board, cable, and audio equipment.

MM/1 SOUND CABLE $10.00 - Connects MM/1 sound port to stereo equipment for recording and palyback.

GNOP $5.00 - Award winning version of PONG(tm) exclusively for the MM/1. You'll fo crazytrying to beat the clock and keep that @#$%& ball in line! Professional pongists everywhere swear by (at) it! Requires MM/1, mouse, and lots of patience.

CoCo Products (DECB)

HOME CONTROL $20.00 - Put your old TRS-80 Color Computer Plug n' Power controller back on the job with your CoCo3! Control up to 256 modules, 99 events! Compatible with X-10 modules.

HI & LO RES JOYSTICK ADAPTER $27.00 - Tandy Hi-Res adapter or no adapter at the flick of a switch! No more plug and unplugging of the joystick!

KEYBOARD CABLE $25.00 - Five foot extender cable for CoCo 2 and 3. Custom lengths available.

MYDOS $15.00 - Customizable, EPROMable DECB enhancement. The commands and options Tandy left out! Supports double sided and 40 track drives, 6ms disk access, set CMP or RGB palettes on power-up, come up in any screen size, Speech and Sound Cartridge support, point and click mouse directory, and MORE OPTIONS than you can shake a stick at! Requires CoCo3 and DECB 2.1.

DOMINATION $18.00 - Multi-Player strategy game. Battle other players armies to take control of the planet. Play on a hi-res map. Become a Planet-Lord today! Requires CoCo3, disk drive, and joystick or mouse.

SMALL GRAFX ETC.

"Y" and "TRI" cables. Special 40 pin male/female end connectors,

priced EACH CONNECTOR -		$6.50
Rainbow 40 wire ribbon cable, per foot -		$1.00
Hitachi 63B09E CPU and socket -		$13.00
MPI Upgrades for all small MPIs (satellite board) -		$10.00
Serial to Parallel Convertor with 64K buffer and external power supply -	**NOW ONLY**	**$28.00!!!**
Serial to Parallel Convertor (no buffer) and external power supply -	**ONLY**	**$18.00!!!**
2400 baud Hayes compatible external modems -		$15.00
Serial to Parallel Convertor or Modem cable (4 pin to 25 pin) -		$5.00

ADD $3.00 S&H FOR FIRST ITEM, $1.00 EACH ADDITIONAL ITEM

SERVICE, PARTS, & HARD TO FIND SOFTWARE WITH COMPLETE DOCUMENTATION AVAILABLE. INKS & REFILL KITS FOR CGP-220, CANON, & HP INK JET PRINTERS, RIBBONS & vr. 6 EPROM FOR CGP-220 PRINTER (BOLD MODE), CUSTOM COLOR PRINTING.

Terry Laraway
41 N.W. Doncee Drive
Bremerton, WA 98311
360-692-5374

The BlackHawk MM/1b

Based on the AT306 board from Kreider Electronics. Features built into the motherboard include:

- 16 bit PC/AT I/O bus with five slots
- MC68306 CPU at 16.67MHz
- 512K to 16MB of RAM with 30 pin SIMMs (4 sockets)
- IDE Hard Drive Interface (2 drives)
- 360K-1.44MB Floppy Drive Interface (2 drives)
- Two 16 byte fast serial ports (up to 115K baud)
- Bi-directional parallel printer port
- Real-time clock
- PC/AT keyboard interface
- Standard PC/AT power connector
- Baby AT size - fits standard PC case
- BASIC (resembles Microsoft BASIC)
- MGR Graphical Windowing Environment with full documentation
- "Personal" OS-9/68000 Vr 3.0 (Industrial with RBF)
- Drivers for Tseng W32i and Trident 8900 VGA cards
- Drivers for Future Domain 1680 and Adaptec AAH15xx SCSI cards
- OS-9/68000 Vr 2.4 with Microware C 3.2, Assembler, MW Basic (like Basic09), MW Debug, MW Programmers Toolkit
- UUCP from Bon Billson
- Ghostscript (software PostScript interpreter)
- Many other utilities and tools

Prices start at $400!
(motherboard, Personal OSK, & MGR only)

BlackHawk
Enterprises, Inc.
756 Gause Blvd. #29
Slidell, LA 70458
E-mail: nimitz@stolY.com

The next element of the System Byte is the 3-bit Interrupt Priority Mask (IPM). This is a very powerful feature of the 68K that allows the processor to control which interrupts can occur. This 3-bit field can have a value from 0 to 7. Level 7 is the highest priority and 0 is the lowest. An IPM of a given value inhibits interrupts of lower or equal priority from occuring. One exception to this rule is that level 7 interrupts can always occur because they are edge-triggered. All other interrupt levels are level-triggered.

The final element in the System Byte is the Trace Bit (T bit 15 above). This feature allows the processor to single-step through a program without having to modify the program's instruction stream. This feature is frequently used by debuggers. The System Byte is not accessible from User Mode.

The User Byte of the Status Register contains five flags (X,N,Z,V,C). These values are accessible from both Supervisor and User Modes. The flags are generally modified when operations are performed on Data Registers and their values are used in conditional branches. These flags are X (eXtend), N (Negative), Z (Zero), V (oVerflow), and C (Carry).

One final register of great importance is the Program Counter or PC. This is a 32-bit register that contains the address of the next instruction to be executed. Although this register is not a general-purpose register, it identifies the place where the currently executing instruction resides and is often used to obtain related read-only information that is built in to the program's image. One very powerful class of addressing modes in the 68K is Program Relative Addressing which is very important for writing Position Independent Code.

Memory organization in the 68K is what is called Big-Endian. This means that the high-order byte of multi-byte values is stored at the first (lowest) address. Examples of other Big-Endian processors are the IBM Mainframes and Midrange, Sun Microsystems SPARC, Zilog Z8000, and the IBM/Motorola PowerPC. Processors that order their bytes the other way (low-order first) are called Little-Endian. Some examples of these are the DEC PDP-11, VAX, and Alpha, the MIPS R-XXXX RISC processors, and the Intel 80X86. Some of the more recent RISC processors can use either format but they all boot up in their "native modes".

Each system has its advantages and disadvantages. The Little-Endian byte ordering is more natural and logical but the Big-Endian method places printed data in an order that those reading from left to right like to see it when using the Arabic system for number representation. When looking at a hex dump of memory data on an IBM PC, for example, the bytes appear backwards and you have to mentally turn them around to understand them. The same hex dump on a 68K machine appears in the correct order. It was after my trip to Israel and Egypt last year that I realized that Arabs and Israelis probably prefer Little-Endian byte ordering. Even though we represent numbers in the same way, we read them in the opposite order because Arabic and Hebrew is read from right to left!

The Powerup Process

.One of the most interesting subjects to me is the sequence of events that happens when the 68000 first comes up after either power on or reset. The original 68000 requires the Reset pin to be held low for a minimum of 100 milliseconds after power is applied to the chip. This ensures that all of the logic on the chip stablizes before execution begins.

The 68000 has a 256-entry Exception Vector Table that begins at location $00000000. Each entry is a 4-byte Long Word. All entries, except the zeroth one, are program address pointers. The first two entries are different from the others and are used just after powerup.

When the processor first comes alive, the Supervisor Bit is set. The first thing that it does is to fetch the first long word from location 0 thru 3 and place it into the SSP (also called the ISP for Interrupt Stack Pointer). In the 68000, this requires two bus cycles because the Data Bus can only fetch two bytes at a time.

Next, the processor fetches the second long word from location 4 thru 7 and places it into the PC. This is the Reset Vector. All of the other 254 Exception Vectors are used after the processor is up and running.

But there's another important reason why these two long words are different from the others. A typical computer contains two kinds of memory: ROM and RAM. ROM is Read Only Memory and is generally non-volatile. RAM is Random Access Memory and is generally volatile.

When power is first applied to the system, the very first memory that is fetched must come from ROM because power has been off and RAM is in an unknown and uninitialized state. The first long word to be fetched becomes the working Stack Pointer value and should come from ROM. The second is loaded into the PC and it must point somewhere into ROM, generally the same ROM it was stored in. The rest of the Exception Table can also point into ROM and often does in embedded systems because all program addresses are frequently pre-determined and burned into a ROM. But general purpose systems need to have the rest of the Exception Vectors stored in RAM because vector addresses are often not known until run time, especially when using loadable device drivers.

This gives us a choice to make: If all Exception Handlers are known and fixed when the system is designed and built, the Boot ROM can be permanently mapped beginning at location 0. This makes it simple but inflexible. If even one Exception Handler has to be determined after boot time, then the table has to be stored in RAM. This means that RAM should be mapped beginning at location 0. In this case we have a problem: The first two long words must come from ROM.

There are at least three ways to solve this problem. Some system designers simply force the first two Long Words to map to the first eight bytes of the boot ROM. If RAM normally starts at zero this makes the first eight bytes of RAM inaccessible. But this is a small sacrifice.

Another approach takes advantage of the fact that the first four accesses

after reset are known to be fetches for the ISP and PC values. This can also be done by simply Shadow Mapping the boot ROM to location 0 for the first four bus cycles. A reset circuit temporarily disables the normal address decoding logic during the first four accesses.

Another common approach is to use the three Function Code pins on the chip. During each access, the chip identifies the cycle as either Program Space or Data Space. The fetches of the first two long words at reset are identified by the processor as being program accesses. Fetches for any of the other vectors in the Exception Vector Table (during exception processing) are identified as being in Data Space. After this reset process, RAM should appear beginning at location 0 while ROM is somewhere in high memory.

After the first two long words have been loaded into the ISP and the PC, normal processing begins and the very first instruction is executed. It is a real thrill to write this very first instruction. After doing it you feel like a real embedded programmer. Now let's write some startup code:

```
;<<<<<<<<<<<<<<<<<<<<<<<<<<
Boot ROM Image Code
>>>>>>>>>>>>>>>>>>>>>>>>>>


        SECTION boot
Reset_ISP:  DC.L    ROM_Size
Reset_PC:   DC.L    Boot
;Boot Parameters
Param1:     DC.W    $4000 ;Interrupt Stack Location

;Reset execution begins here
Boot:       MOVE.W  (Param1,PC),A7
            OR      #$0700,SR
```

In the above example, Reset_ISP is a symbol that is linked to the very first byte in the Boot ROM. Earlier I said that the long word stored beginning at this location is loaded into the ISP at Reset. But what is stored there is a value called ROM_Size.

Why am I doing this? Actually, I can load a new ISP value later on any time I want to. As long as I don't call a subroutine or do anything else that uses the stack pointer, it doesn't matter at all that the value was loaded into the ISP before the processor started to run.

I always use these first four bytes to record the number of bytes used in the Boot ROM. This is so that I can easily tell how long the Boot Image is that is stored in the Boot ROM. I will explain why I do this later when I explain how the PREBOOT process works.

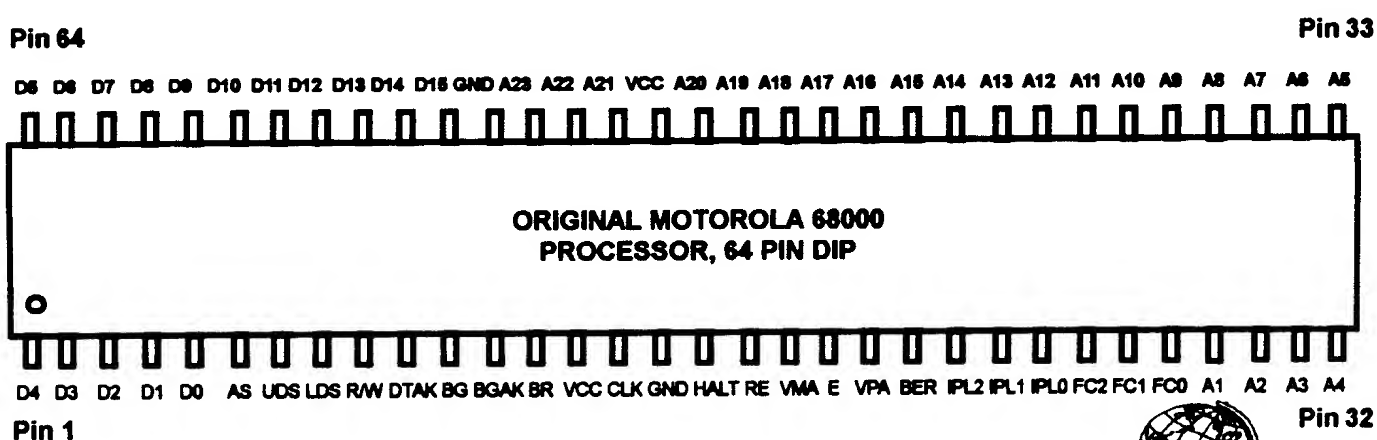
Anyway, after these two values are loaded, program execution begins at the location labeled Boot. Here is my very first instruction. This instruction loads a new ISP value. But I am loading it from an earlier place in the Boot ROM using Program Relative Addressing. This makes this instruction Position Independent. Also notice that I am loading a 32-bit Address Register with a 16-bit Short Address. This instruction loads the 32-bit Signed Extended value of $00004000 into the ISP.

The first instruction labeled Boot can be located anywhere in 32-bit address space. I like to reserve the first several byte locations in the Boot ROM to store pre-determined Boot Parameters. Since the program execution sequence can take many instructions of indeterminate length, I have the Reset Vector point past this table of Boot Parameters. Now looking at the ROM with either a software or hardware tool, I can easily view and even change them.

We'll find out later why the Boot Image that is stored in the Boot ROM can actually be a changable image in RAM or on disk. It will become apparent why we keep the Boot Parameters handy up at the front of the Boot Image.

In the next article we'll take a deeper look at the advanced 68K interrupt structure. If you have any comments or requests, please feel free to write me at either <gecko@onramp.net> or at the address given below:

Paul K. McKneely
technoVenture, Inc.
P. O. Box 5641
Pasadena, Texas 77508-5641



What are you waiting for?

Get your friends to subscribe to the only magazine that still supports the Tandy Color Computer...

"the world of 68' micros"!!

The more people who want the support, the longer it will be here!

What Happened to Burke & Burke?

Chris Burke

Burke & Burke is no longer in business, but Trisha and I are still together and living in Washington state. I still develop custom entertainment products (but not for the Color Computer) through my new venture, Serotonin Software.

You can see some of my handiwork in the Super Nintendo (SNES) version of Sinistar, one of the five classic arcade games hand translated from the original 6809 code on the Williams Arcade Greatest Hits cartridge from Williams Entertainment and Digital Eclipse Software.

I can't provide technical support for old Burke & Burke products, but I still have some inventory and own the distribution rights for all but a few.

In honor of the 1997 Chicago CoCoFest, I've re-released several familiar Burke & Burke products as shareware. Also, never before available, you'll find the source code and schematics for the popular CoCoXT and CoCoXT-RTC hard disk interfaces. The shareware disks include text versions of the manuals to make distribution easier. If you value these products, even after so many years, you can send the shareware fee to me at Serotonin Software. Glenside Color Computer Club will handle distribution.

Send requests for disks along with $3 shipping and handling to:
B&B Software
c/o Glenside Color Computer Club
119 Adobe Circle
Carpentersville, IL 60110-1101

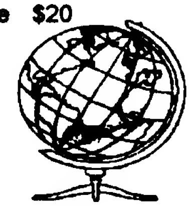
You may also send e-mail to:
tonypodraza@juno.com
for additional information.

Registrations fees should be sent to:
Serotonin Software
P.O. Box 1045
Woodinville, WA 98072-1045

Orders for shareware **WILL NOT BE PROCESSED** at the Serotonin address!!

Burke & Burke shareware pricing:

Daggorpatch	$5
ZClock	$5
World Class Chess	$5
SCSI-512	$5
(this is by B&B, NOT the Matt Thompson Software!)	
EzGen	$5
Pertascii	$5
Wild & MV	$5
XT-ROM	$10
(code only, not burned in ROM)	
File Recovery System	$10
RSB	$15
File System Repack	$15
PowerBoost	$15
CoCoXT Source Code	$20
EzGen Source Code	$20
Pertascii Source Code	$20



Announcing Nitro Level III!

How many times have you been unable to load a driver due to not enough system RAM? How would you like to have up to 32K of system RAM available? How is this possible?

In effect, Nitro Level III turns the system into a Kernel only process, with 48K or RAM, and 2 IO processes (RBF and SCF), each with 16K of RAM. This is similar to Grfdrv having it's own 64k memory area.

The kernel process contains the minimum modules to run an OS-9 system, and also the descriptors. The RBF/SCF processes contain the IO modules, and the IO buffers.

There are 2 big benefits here:

1 - Both RBF and SCF are not in system memory at the same time, so you save RAM.

2 - You don't have 16K of SCF or RBF modules, so everything up to 16K can be used as device data storage (sector buffers, etc.)

Level III works only with Nitro (all versions). It can be purchased from FARNA Systems alone ($20) or with the latest version of Nitro ($45 for Nitro v2.00 and Level III). See the FARNA ad in this issue for ordering information.

SOURCES!

I would really like to run this as a regular column. What I am looking for is sources for hard to find and bargain items for CoCo, 68K, and general computer use. If you find a treasure trove of good, inexpensive parts, let me know!

5.25" 360K and 3.5" 720K Double Density Disks

These are getting harder to find locally! Radio Shack has them, but at a hefty price of $10 per box! These guys have 5.25" double sided, double density disks (also used in single sided drives) at $8.00 per 100 relabeled (used but tested good), or $8.50 for 50 new. 3.5" 720K disks are $12.50 for 50 or $24 for 100 (all new). These are UNFORMATTED prices. They can be purchased for a few dollars more preformatted for IBM compatibles. Call for prices on 1.4M disks.

Media Source
2197 Canton Rd.
Suite 210
Marietta, GA 30066

Orders: 800-241-8857
FAX: 770-919-9228
Bus Phone: 770-919-0059
E-Mail sales@mediasource.com

Contributed by James H. Kirby <jkirby@mail.oeonline.com>

Easy Disk Drive with Case

Roger Merchberger

Convert an old IBM external drive for CoCo use!

This is a short article about modifying an IBM external 5.25" 360K floppy drive to use it with your Tandy Color Computer (any model). These drives are extremely well built (read: tanks), are of very high quality. Like everything else IBM, when the drives debuted, I'm sure they were extremely expensive as well, but now they can be found for quite a reasonable price.

If you get a complete drive (cable and all) the first thing you might think is that IBM went non-standard all the way, like usual. That's only half-true. IBM chose a 37-pin d-sub connector for the cable so they could easily supply all the necessary signals with an ample supply of ground lines (half the lines on a 44 pin connector are grounds). Many laptops still use this connector for external floppy drives.

The dimensions of this drive are 16" deep by 9" wide by 2.5 inches deep, and one would think they're made of armor plate due to the weight. The weight comes from the total RFI metal shielding around the power supply and floppy drive, and a rather high-capacity power supply, as well. The 3.5" 720K model is similar, but smaller. There may be some other differences between the 5.25" and 3.5" drives.

Here are the step-by-step instructions to modify the drive:

Step 1: Disassemble the case. You will find 8 screws in the bottom of the case, 6 of which are standard phillips, and the other two appear to be Torx screws, but with a post in the middle of the hole. Your best bet (and what I did) is to bend the post out of the way, jam a regular screwdriver that you wouldn't mind to get screwed up into two of the six points of the star that fit the best around the bent post, and don't bother to put them back in. This was, of course, before I owned a Dremel tool, and one could grind the post out and use a Torx driver, but that sounds too much like work to me. Once these two goofy screws are out of the way, you shouldn't find any more nonstandard screws. Once the case is opened, you need to remove the shielding from both the power supply and the floppy drive, to remove the nonstandard cable.

Step 2. Once the cable is removed, you need to fit a standard CoCo floppy cable in the drive. There are several ways to do this, but I've only tried one. One way would be to thread a standard CoCo cable along the same route as the old one. If this will be your only drive, that may work. If you wish to have more than one drive, this most probably will not work. In this case, you may have to go by Radio Shack and get a length of 40 conductor ribbon cable and three 40 pin crimp-on edge card connectors. Crimp one end on the cable and press it on the floppy drive. Run it along the path of the old cable and out the case. Leave enough cable to attach your other drive and then extend to the controller. Crimp a connector on for the other drive and on the end for the controller. You can use two of the IBM drives by running a double length of cable in the second drive. I have seen drives run with three to four feet of ribbon cable with no problems.

What I did was this: When looking at the top front of the drive, measure 11" back and 2" in from the left, and cut a hole 2" wide and .75" back in the drive. That's the size of the hole I made, but be sure to make allowances for a little extra room if necessary to fit your cable. I cut this hole drilling holes in the corners, and using a saber saw to cut the holes. A Dremel tool with a cutting blade would be ideal for this job. The hole should be very near the floppy cable connector and directly behind the rear EFI shield for the floppy.

Here comes the fun part: you need to make a custom cable from standard IBM parts. Remember, all standard floppy cables have 34 pins. Go purchase an IBM floppy cable that's designed to add an extra floppy port for attaching a floppy-based tape drive. This cable usually has three connectors, but only two are necessary: one is a MALE pin connector, and a standard older card edge connector. Also purchase one of those little 3.5" floppy adapters that change from a pin connector to the older card edge connector.

Fit the floppy cable card-edge connector through the hole, and in the shielding onto the floppy connector. Then carefully screw down the shielding, securing the cable (be careful not to cut the ribbon cable... the metal is not extremely sharp, but it can cut the insulation if you're not gentle...) then put the case back on. Pu' the 3.5" to 5.25" floppy connector in the male pin connector, and fit this into one of the connectors on your standard Tandy cable.

What this does is give you a "base" floppy, that a FD-500, FD-501 or FD-502 sits on top of quite nicely. If you have two drives in your FD-50x, you'll need to crimp on one more connector to the cable, to make the extra spot for the IBM cable. This also means you need to split the outer casing on the cable if you have a round floppy cable so you can attach an extra connector. The round cable has a flat ribbon cable rolled up inside and some extra shielding. You can easily split the case far enough back to fit an extra connector to fit the IBM cable up from the IBM drive.

One last thing: It is best to leave the terminating resistor on the IBM drive, as it is the least accessible drive of the pack. Make it the last drive on the floppy chain. This also insures that you have a compatible 5.25" floppy available for transferring that old software, and now you car modify your FD-50x for one or two 3.5' 72K floppies for OS-9!

The floppy article in the July/August 1996 issue said that the FD-502 power supply is a little weak. I have had the standard Drive 0 and a 1.4Meg 3.5" floppy (jumpered for 720K only operation) running without the fan with no problems for over 3 years now. The 3.5" drives take much less power, and run just fine!

At the time of this writing, the IBM cases (without floppy) were for sale at B.G. Micro for $10.00. Add $10.00 for a floppy, and some hardware hacking, and voila! A wonderful floppy drive for your CoCo.

If anyone has questions about this modification, or anything else that I might be able to help you with, feel free to e-mail me at my Internet address: zmerch@ northernway.net.

Year 2000...

Robert Gault

Are you ready, and is your CoCo?

Just in case you have been off on a desert island, when the year 2000 rolls around all hell will break loose in computer land. Why is that? Most computer systems and software store only the last two digits of a year in the date information. On January 1, 2000 your computer may represent the date as Jan. 1, 1900. Some systems will not give the wrong date. They will lock up and cease to function!

The Gartner Group, Inc., an information technology research firm, has estimated that it will cost between $300 billion to $600 billion to correct the Year 2000 problem worldwide1. Every program that uses a six digit ASCII date field (mm/dd/yy) must be searched for each occurrence of the field and patched. Cost estimates vary, but to correct source code, $1.10 per line is typical.

Many senior executives are unaware of the problem. Many others don't understand it or don't believe it to be serious. Worse still, many Information Support engineers when asked say, "It's no problem for me, I intend to retire in 1999." If you think this is silly, consider this. Your new car five year warranty, which you just bought, may already have expired; 1996+5= 2001=1901 on a typical system. Think of your bank accounts, stock holdings, social security— all managed by computer systems.

Where do you stand with your Coco? If you have never gone past Disk Extended Basic you may be in good shape. I say may, because Disk Basic does not date disk files. Are you using any programs which incorporate dates? Better check them.

If you have moved up to OS-9, you will have problems. Just how bad they will be depends whether you use your Coco for fun or business. Let's see where OS-9 makes use of dates.

Each disk used in OS-9 has a creation date stored in logical sector zero (LSN0) in five bytes; y:m:d:h:m. That means there is only enough room for the last two digits of the year. Each file descriptor contains two dates, creation and last modification; again using five bytes. Is this cosmetic, or will RBF (random block file manager) choke on a file which has a modification date of 00 with a creation date of 96?

At the command level Date and Setime will not work correctly as the system does not leave enough room on the system direct page. There is only one byte for the year. A friend of mine has decided to patch Date to replace the hard coded "19" with "20." He intends to switch to the new Date after the year 2000 but is that enough? The command "dir e" will let you access your files' creation dates but only displays two digits for the year. Suppose you need to sort your files by date, what then?

Any software that makes use of dates in OS-9 can be no better than the system. Do you use any software that automatically inserts the current date? Will your SmartWatch(r) or other hardware clock save you? No, it won't — it probably does not yield four digit years (the SmartWatch does not) and if it did, OS-9 can't use the information. Something to think about, isn't it?

For the experimenter, Date can be patched by looking for the data string, $31B9. This should be changed to $32B0 and the CRC updated to make Date work correctly after 1/1/2000. The stock version of Date has "19" located at the front of the module along with other ASCII data.

For more information, check out this internet web site:

http://www.gartner.com/aboutgg/pressrel/pry2000.html

I can be reached in care of this magazine or via internet at

robert.gault@worldnet.att.com

Disk EDTASM Modification

Modify Edtasm to display on 40 or 80 column screens

Here is a BASIC program that patches the original EDTASM floppy to work on 40/80 column screens. After patching, you'll need to put the CoCo into the 40/80 column screen, BEFORE running the "DOS.BAS" program on your EDTASM disk. A tiny one-liner called "E.BAS" sets the screen width and palettes to your own personal preferences, then runs DOS.BAS. The program E.BAS looks like this:

```
10 WIDTH80:PALETTE0,0:PALETTE
8,63: ATTR 0,0:CLS1:RUN"DOS.BAS"
```

This sets 80 column, white text on black background, and runs DOS.BAS, listed below:

```
10 A$=3DHEX$(PEEK(&H0FFFE))+
HEX$(PEEK(&HFFFF))
20 IF A$<>"8C1B" THEN
CLS:PRINT"PATCH ONLY FOR COCO
III":END
30 POKE &H9692,17=7F
40 PCLEAR 16
50 POKE &H9692,9
60 PALETTE 12,63
70 PALETTE 13,0
80 WIDTH 32:CLS:VERIFY ON
90 IF FREE(PEEK(&H95A))<7 THEN
PRINT"DISK IS TOO FULL":END
100 PRINT"PATCHES FOR EDTASM TO
RUN"
110 PRINT
120 PRINT
130 PRINT"INSERT COPY OF
EDTASM"
140 PRINT"PRESS ENTER WHEN
READY"
150 A$=3DINKEY$:IF A$<>CHR$(13)
THEN 150
160 PRINT"LOADING EDTASM"
170 RENAME"EDTASM.BIN" TO
"EDTASM.OLD"
180 LOADM"EDTASM.OLD"
190 PRINT"PATCHING..."
200 READ AD$,DT$
210 IF AD$=3D"END" THEN 240
220 POKE
VAL("&H"+AD$),VAL("&H"+DT$)
230 GOTO 200
240 PRINT"SAVING..."
250 SAVEM"EDTASM.BIN",&H1600,
&H4A7F, &H1600
260 PRINT"DONE."
270 PCLEAR 4:CLEAR
200,&H7FFF:NEW
280 DATA 1617,84,1643,31,1D18,7F,
1D19,FF
290 DATA 1D1A,DE,1D1B,6E,1D1C,9F,
1D1D,FF
300 DATA 1D1E,FE,1D1F,12,1D20,12,
1D21,12
310 DATA 1D22,12,1D23,12,1D3F,BD,
1D40,A1,1D41,B1,1D42,12,1D7A,10,23B8,31
320 DATA END,END
```

CoCo3 Extended Memory Secrets Part 2

Herbert Enzman

Practical use of CoCo3 Video

Introduction

In PART 2 of this series, I'll cover the practical use of COCO 3 video. As in PART 1, some of this information might be know by you, but not to others, so some basics first. We will cover mostly the 80 column screen, with attributes. The NON-attribute screen will be covered in PART 3.

Screen Memory Useage

Super ECB (SECB) has reserved block $36 for use of the screen memory, and maps it into $FFA1 ($2000 - $3FFF) when it needs to display the screen. In the attribute mode, each screen LINE uses 160 bytes (80 for characters and 80 for attributes). Since there are normally 24 lines per screen, then 160 X 24 = 3840 bytes used by the screen. Since each BLOCK is 8K in size, you can see that less than half of block $36 is used for the screen.

The screen runs from $2000 - $2EFF, with $2F00 - $3FFF unused by SECB (how to use that area later). When writing DIRECTLY to the 80 column screen, you must remember to send characters to the EVEN address and the attribute byte to the ODD address, or some very strange results will occur. SECB takes care of this for you when you use it's "character out" routine, but now it is up to you. TABLE 5 is a quick reference for which HEX digit is ODD and which is EVEN.

$FFB0 - $FFBF are used for the palette registers. $FFB0 - $FFB7 are reserved for BACKGROUND colors, and $FFB8 - $FFBF are reserved for the FOREGROUND colors. You can set-up each register with any color (from $00 thru $3F) that you like.

SECB keeps 3 tables in memory for the palette registers, as shown by TABLE 7. The SECB MAIN table is used to reset the palette registers on a hardware RESET, so if you don't want the colors to change upon RESET, you should also set the colors of your choice in that table also. The other two tables are used for cold start or the commands 'CMP' or 'RGB', they are the 'default' tables.

DISK EDTASM uses $FFB8 for it's foreground and $FFB0 for it's background; E/A 6309 uses $FFB8 for foreground and $FFB4 for background. Go ahead and use Z-BUG to change these registers for the colors of your choice. Standard DISK EDTASM users will want to set up an 80 column screen (listing 2 from part 1) first.

Attributes...

Now it is time to cover the ATTRIBUTE byte. TABLE 6 shows it's format. To help keep down confusion when calculating the value of the attribute byte, I recommend this format: XX XXX XXX. Start with bit 7 first and work your way to the right, as in this example: Let's say you want a yellow character on a black background that flashes; and The color yellow is located in palette register $FFBD and black in $FFB6. You would first set bit 7 to a 1 for flash (1X XXX XXX). Since you don't want underlining, bit 6 is a 0 (10 XXX XXX). Now you look at TABLE 4 for the foreground color (yellow is in $FFBD), find it's BIN code and insert it into the byte (10 101 XXX). Next comes the background color (black in $FFB6), and put it's BIN code into the byte (10 101 110). Now all you have to do is to convert it into HEX (1010 1110 = $AE) and you have the attribute byte that needs to be sent to the screen with each character.

You don't have to use the same attribute for each character if you don't want to. You could change the attribute byte for each character sent, but that would be quite confusing. Generally you would want to use the same byte, at least for the same line of text. But you CAN mix 8 foreground colors and 8 background colors on the SAME screen when you WRITE DIRECTLY to the screen.

With SECB you are stuck with two colors. $FE08 is SECB's 'current attribute temp'. You can change it when using the CHROUT routine to change the attribute, but it would be much easier just writing directly to the screen. You can experiment with this temp with Z-BUG's 'slash' command, to see what happens. TABLE 7 shows what I have found so far for SECB's screen routine temps.

The "screen grids" in the COCO 3 manual are a little small for quick use, so I would recommend that you tape several pieces of paper together and make yourself a larger "grid". Make each grid square 1/4 by 3/8 inches in size, so that you can write the address into each. Make it similar to the grid on page 284, 80x24, then number each grid square with an EVEN address (just for characters) to keep down confusion. Number each grid square with this format: $0000,$0002,$0004, ending with $0EFE. The reason for using a 0 in the first digit instead of a 2 is because block $36 can be mapped into ANY $FFAx register. It now is an offset to be added to the address range of that block.

For example: let's say that you have mapped block $36 into $FFA3/$FFAB ($6000 - $7FFF). The screen would then start at $6000 instead of $2000, so adding 0xxx to $6000 would give the proper address. In other words, all you have to change is the 1st digit, when you map block $36 into a different $FFAx register. I know that it will be a boring job to make this grid (I did it), but it will speed up finding screen locations in the long run.

Practice makes perfect!

Now that you have some information on screen use, it is time for some practice. DISK EDTASM users will want to set up the 80 column screen, if you haven't done so yet. Now enter Z-BUG in byte mode, and change the $FFBx registers to the colors that you desir[e] Now, change $E0E4 to $36 (remember thi[s] from PART 1?) to map block $36 into the range of $8000 - $7FFF. Next, clear the screen with the "CLEAR" key and do "6EC0 / " and put $41 there. You should see an "A" pop up in the lower part of the screen. Now do "6EC1 /" and put $AE there, and you should see a flashing "A" with the colors that you set up into $FFBD and $FFB6. Set $6EC1 to $EE and you will see an underlined flashing "A". $6E will give you just an underlined "A" and $2E will give you a steady "A".

Now experiment with various other values at $6EC1 to see how the attribute byte works. Once the cursor gets down towards the "A", you may have to clear the screen again to stop the "A" from scrolling out of it's position.

E/A 6309 users will have to use LISTING 3 from PART 1 to set block $36 into $FFAB / $E0EC for this experiment (remember from part 1 as to why?). If E/A 6309 crashes when using $FFAB, then you will have to switch to using $FFAC. As stated before, I am using the FIRST version of E/A 6309, and the program ends at $54DF, and I understand that subsequent versions are different. I don't know if the program is longer and ends in the $6000 address range or not, the reason for this warning.

This experiment demostrates severa[l] things: 1) you can map block $36 where ever you like and SECB will still display it. 2) what the various values in the attribute byte will display. 3) more practice on block switching. 4) the use of the $FFBx registers and 5) that you can have a multi-colored screen when writing directly to it.

Listing explanations

LISTINGS 5,6,7 are short demo programs that you can use to further practice writing to the screen. 6 and 7 are the MAIN body programs, and 5 is the subroutine that does most of the work.

I have set LISTING 5 up so that it has 3 entry points for 3 purposes. It will work with both DISK EDTASM and E/A 6309 (the only difference is the $FFAx register as noted in the comment column). The two instructions "FCB $XX" and "LDB #$XX" are where you put the attribute of your choice where the XX's are. If you enter the subroutine at "STRING", the B register will be loaded with the attribute that PRECEEDS the text string, as you can see by the "LDB -1,X" instruction. If you want to use the same text, but with a different attribute, then PRELOAD the B register with the attribute and enter the routine at "SCRIPT". I[f] you are just sending one character to th[e] screen, then PRELOAD the registers X,Y and B and enter at "SCREEN". I use a negative 'stop' character in the text line (FCB 'E+$80

in the listing) so if you add more text, do the same or the routine will just WIZ thru memory until it finds a negative character. You will notice that the routine swaps the block at the $FFAx registers and not the $E0Ex registers because we are writing *directly* to the screen and not going SECB's routine. This is an important thing to remember. LISTING 6 will get the attribute byte that preceeds the text string; and LISTING 7 will display the same string with a different attribute by loading the 'B' register ahead of time.

To use the other half of the screen block, you will have to change the pointers in SECB to direct it to the 2nd half. LISTING 8 is a program that will demonstrate the use of the second half. DISK EDTASM users should NOT use the code with "*", it is for E/A 6309 users only! Type it in, assemble and run it. The cursor will disappear, and it will seem like the program has crashed, but it hasn't. The cursor is now located in the 2nd half of screen block, out of your view. You will have to type BLIND, so do this next step CAREFULLY. You are still in Z-BUG, so type "B" for 'byte mode'; then type: "FF9D /" and change $FF9D to $DA. You will now be in the 2nd half of the screen block, and there is the cursor, blinking happily.

To get back to the 1st half of the screen block, change $FF9D to $D8. I will get into more detail on $FF9D in the next installment. For now, to toggle between the two screens, just set $FF9D between $DA for the 2nd half and $D8 for the 1st half. The 1st screen is: $2000 - $2EFF and the 2nd = $3000 - $3EFF. With some imagination, you can merge this information and have plenty to play with until next time

Part 3 will be...

Next time I will discuss what I have found with the NON-attribute screens and the other $FF9X registers. This information is pretty interesting and with some imagination, can be very usefull.

TABLE 4

HEX FORE	HEX BIN	HEX BACK
$FFB8	000	$FFB0
$FFB9	001	$FFB1
$FFBA	010	$FFB2
$FFBB	011	$FFB3
$FFBC	100	$FFB4
$FFBD	101	$FFB5
$FFBE	110	$FFB6
$FFBF	111	$FFB7

TABLE 5

ODD	EVEN
00	01
02	03
04	05
06	07
08	09
0A	0B
0C	0D
0E	0F

TABLE 6

BIT	USAGE
7	1= blink
6	1= underline
5	*
4	* foreground (B8 - BF)
3	*
2	#
1	# background (B0 - B7)
0	#

****

TABLE 7 - Palette register tables

ALL addresses HEX

HARD WARE	SECB MAIN	CMP SETUP	RGB SETUP
FFB0	E678	E654	E664
FFB1	E679	E655	E665
FFB2	E67A	E656	E666
FFB3	E67B	E657	E667
FFB4	E67C	E658	E668
FFB5	E67D	E659	E669
FFB6	E67E	E65A	E66A
FFB7	E67F	E65B	E66B
FFB8	E680	E65C	E66C
FFB9	E681	E65D	E66D
FFBA	E682	E65E	E66E
FFBB	E683	E65F	E66F
FFBC	E684	E660	E670
FFBD	E685	E661	E671
FFBE	E686	E662	E672
FFBF	E687	E663	E673

SECB SCREEN TEMPS

$FE00/01	Cursor location
$FE02	working char. count
$FE03	working line count
$FE04	# char. per line
$FE05	# lines per screen
$FE06/07	screen end location
$FE08	current attribute
$FE09	unused
$FE0A	foreground color
$FE0B	background color

LISTING 5 - Write to screen routine

UPON ENTRY: X = Points to text that is to be printed to screen. Y = location on screen to put text. B = SEE TUTORIAL TEXT

```
STRING LDB   -1,X
get attribute that is before text string
SCRIPT LDA   ,X+      get text character
       PSHS  A
save for stop character test
       BSR   FIX      send it to the screen
       TST   ,S+
was character a STOP char. (negative)?
       BPL   SCRIPT
no, loop for more characters
       RTS            DONE - return
FIX    ANDA  #$7F     drop MSB first
SCREEN PSHS  Dsave for after block swap
       ORCC  #$50     disable interupts
       LDA   #$36
= BLOCK # that screen uses
       LDB   $FFA3    *** $FFAB for 6309 users
       STB   SAVE
save current block # for return
       STA   $FFA3    *** $FFAB
       PULS  D
. character and it's attribute
       STD   ,Y++     store both to the screen
       LDB   SAVE
get original block # that was saved
       STB   $FFA3    *** $FFAB
       ANDCC #$AF     enable interupts
       RTS            return for more characters

SAVE   RMB   1 ** temp for current block #

       FCB   XX
PUT desired attribute here in place of 'XX'
TEXT1  FCC   /YOUR MESSAGE HER/
       FCB   'E+$80
this is for STOP printing code
       END
```

LISTING 6

```
GO     LEAX  TEXT1,PCR
       LDY   #$2AF0
       BSR   STRING
       SWI
```

LISTING 7

```
GO     LEAX  TEXT1,PCR
LDY    #$2AF0
LDB    #$XX   = attribute
BSR    SCRIPT
SWI
```

* LISTINGS 6 AND 7 call LISTING 5. — SEE TEXT
*

LISTING 8 "Second" screen demo

***** CODE with "*" is for E/A 6309 users only!!!!!!!

```
GO     NOP
*      CLR   $FF91    set TR=0
       LDA   #$3F
end address of second screen (msb)
       STA   $FE06    set it for SECB
       STA   $F688    set it for SECB
       DECA           'A' now = $3E
       STA   $F875    set SECB
       LDA   #$30
       STA   $F7BC
set start address of second screen (msb)
       STA   $F68D    set SECB
       STA   $F6A3    set SECB
       STA   $F6D5    set SECB
       JSR   $F679
Now set up 80 column screen
*      LDA   #1       ###
*      STA   $FF91    ### set TR=1
       LDD   #$3600   set screen colors
       STA   $FFB8    set foreground to yellow
       STB   $FFB0
set background to black  (disk edtasm)
*      STB   $FFB4
set background to black  (e/a 6309)
       STB   $FF9A
set border to black
       SWI            FINISHED
       END
```

ADVERTISER'S INDEX

Color Computer / OS-9 Show & Sale Information

August 2 & 3, 1997
(Sat. 10am-5pm; Sun. 10am-3:30pm)

at the

EMBERS INN

1700 Harrisburg Pike
Carlisle, PA

Overnight room rate: $60
Be sure to ask for the "FEST"rate!

Exit 16 off of the PA Turnpike I-76
Turn to Harrisburg, go 1.3 miles, it's on the right.
OR I-81 Exit 17, turn left, go 1/10th mile, on the right.

There is even more information on Ron Bull's Web site! www.geocities.com/SiliconValley/Vista/1412/BullsBarn.html

Vendors who plan to attend:

Elite Software	SubEtha Software
PA Online	R.C.Smith
Carl Boll	FARNA Systems
StrongWare	Rick Cooper-CFDM
SBug	Adventure Survivors
Bargeman Research Labs	Unlimited Electronics Repair
MonkWare	Paul W. Zibaila III
CoNect	Black Hawk Enterprises
Alan Dages	Glenside CoCo Club

Call 1-717-243-1717 OR 1-800-692-7315 OR Fax (717) 243-6648 for reservations! Limited supply of rooms reserved for the show. Rooms will be released on July 1 and will NOT be available at the show rate!
ADMISSION: $5.00 per person per day or $7.00 for both days (paid in advance). Children under 10 accompanied by a responsible adult are free!

For further information, general or exhibitor, contact:

Ron Bull
115 Ann Street
Duncannon, PA 17020-1204
(717) 834-4314
OR Email me: ronbull@aol.com

FEATURED GUESTS:

Steve Bjork - Bring your ZAXXON and Gwana Bwana manuals - he said he would autograph them for you! Will be giving a seminar on game programming in general.

Kevin Darling - Will enlighten us with "OS-9 and Multimedia"

Marty Goodman - may do a seminar! One of the most infamous CoCo personalities still with us... or is that *the* most infamous of us all???

For a FREE PA State map and Visitor's Guide call 1-800-VISIT-PA - It's FREE!